НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
*Кафедра автоматизованих систем обробки інформації і управління*

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

# Дипломний проєкт
**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**
**спеціальності 122 «Комп'ютерні науки та інформаційні технології»**

на тему: «*Інформаційна система моніторингу та аналізу біржових ринків*»

**Виконав:** студент *IV* курсу, групи *ІС-61*

*Проскурка Даниіл Миколайович* _____
(прізвище, ім'я, по батькові) (підпис)

**Керівник** *доц.,к.ф-м.н. Клименко Олена Миколаївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

**Консультант з графічної документації** *доц., к.т.н., доц. Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

**Рецензент** *доц., к.т.н., доц. Лісовиченко Олег Іванович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

# З А В Д А Н Н Я
**на дипломний проєкт студенту**

*Проскурці Даниілу Миколайовичу*
(прізвище, ім’я, по батькові)

1. Тема проєкту «Інформаційна система моніторингу та аналізу біржових ринків»
керівник проєкту *Клименко Олена Миколаївна, к. ф-м.*
( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
*Технічне завдання*

4. Зміст пояснювальної записки
*1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі*
*2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних*
*3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання*
*4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів*

*5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту*

5. Перелік графічного матеріалу

*1. Схема бази даних*

6. Консультанти розділів проєкту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | *Вивчення рекомендованої літератури* | 07.03.2020 | |
| 2. | *Аналіз існуючих методів розв'язання задачі* | 12.03.2020 | |
| 3. | *Постановка та формалізація задачі* | 20.03.2020 | |
| 4. | *Розробка інформаційного забезпечення* | 02.04.2020 | |
| 5. | *Алгоритмізація задачі* | 17.04.2020 | |
| 6. | *Обґрунтування використовуваних технічних засобів* | 29.04.2020 | |
| 7. | *Розробка програмного забезпечення* | 01.05.2020 | |
| 8. | *Налагодження програми* | 10.05.2020 | |
| 9. | *Виконання графічних документів* | 11.05.2020 | |
| 10. | *Оформлення пояснювальної записки* | 12.05.2020 | |
| 11. | *Подання ДП на попередній захист* | *15.05.2020* | |
| 12. | *Подання ДП на основний захист* | *01.06.2020* | |
| 13. | *Подання ДП рецензенту* | *02.06.2020* | |

Студент Даниіл МИКОЛАЙОВИЧ

Керівник Олена КЛИМЕНКО

| №з/п | Формат | Позначення | Найменування | Кількість листів | Примітка |
|---|---|---|---|---|---|
| 1 | А4 | | Завдання на дипломний проєкт | 2 | |
| 2 | А4 | ДП 6123.00.000 ПЗ | Пояснювальна записка | 46 | |
| 3 | А4 | ДП 6123.01.000 ТЗ | Технічне завдання | 9 | |
| 4 | А3 | ДП 6123.02.000 СБД | Схема бази даних | 1 | |
| 5 | А4 | ДП 6123.03.000 ССК | Схема структурна класів програмного | 1 | |
| | | | забезпечення | | |
| 5 | А4 | ДП 6123.04.000 ССП | Схема структурна послідовності | 1 | |

| Зм. | Арк. | ПІБ | Підп. | Дата | ДП 6123 00.000.00 | | |
|---|---|---|---|---|---|---|---|
| Розробн. | | Проскурка ДМ | | | Відомість дипломного проєкту | Літ. | Лист 1 / Листів 1 |
| Керівн. | | Клименко О. М. | | | | | |
| Консульт. | | | | | | КПІ ім. Ігоря Сікорського<br>Каф. АСОІУ<br>Гр. ІС-61 | |
| Н/контр. | | Тєлишива ТО | | | | | |
| В.о.зав.каф. | | Павлов О.А. | | | | | |

# Пояснювальна записка
# до дипломного проєкту

на тему: Інформаційна система моніторингу та аналізу біржових ринків

Київ – 2020 року

# АНОТАЦІЯ

**Структура та обсяг роботи.** Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 18 рисунків, 12 таблиць, 2 додатків, 9 джерел.

Дипломний проєкт присвячений розробці інформаційної системи моніторингу та аналізу біржових ринків. Метою даної інформаційної системи є підвищення прибутковості від інвестування шляхом пошуку менш ризикованих способів та спостереженням змін стану ринку, на основі яких створювати рекомендації щодо продажу чи купівлі цінних паперів.

Для досягнення поставленої мети необхідно вирішити такі задачі: ведення аккаунту користувача, створення портфелю користувача, збір даних щодо стану ринку, обробка даних з ринку, створення моделі, що відображає ринок, формування звітності.

У розділі інформаційного забезпечення були визначені вхідні та вихідні дані, описана структура бази даних.

Розділ математичного забезпечення присвячений розробці моделі прогнозування.

Програмне забезпечення включає діаграми послідовності, класів, розгортання, специфікацію функцій. Також представлено опис засобів розробки та вигляд звітів системи.

У технологічному розділі наведено керівництво користувача, опис випробувань системи.

ПРОГНОЗУВАННЯ, ФОНДОВИЙ РИНОК, РЕГРЕСІЙНА МОДЕЛЬ, ЧАСОВИЙ РЯД, ТЕХНІЧНИЙ АНАЛІЗ, ФУНДАМЕНТАЛЬНИЙ АНАЛІЗ

# ABSTRACT

**Structure and scope of work**. The explanatory note of the diploma project consists of five sections, contains 20 drawings, 12 tables, 2 applications, 9 sources.

The diploma project is devoted to the development of an information system for monitoring and analysis of stock markets. The purpose of this information system is to increase the return on investment by finding less risky ways and monitoring changes in the market, on the basis of which to create recommendations for the sale or purchase of securities.

To achieve this goal it is necessary to solve the following tasks: maintaining a user account, creating a user portfolio, collecting data on the market, processing data from the market, creating a model that reflects the market, reporting.

In the section of information support the input and output data were defined, the structure of the database was described.

The section of mathematical software is devoted to the development of a forecasting model

The software includes sequence diagrams, classes, deployments, function specifications. There is also a description of the development tools and the type of system reports.

The technological section provides a user manual, a description of system tests.

FORECASTING, STOCK MARKET, REGRESSION MODEL, TIME SERIES, TECHNICAL ANALYSIS, FUNDAMENTAL ANALYSIS

# ЗМІСТ

# ВСТУП

Існує постійний масовий інтерес до того, як фондовий ринок веде себе за різних обставин, таких як: рецесії, економічні кризи, війни, санкції та інші події. Ще на початку початку 20 століття тема ринкових настроїв породила новий розрив між тими, хто залишився скептично налаштованим та тими, хто визнав це важливою концепцією майбутнього ринків.[1] З розвитком IT та вивченням Big Data, як інструменту, з'явилося багато більш широких способів дослідити, який саме вплив на прогнозування ринкових цін може бути. Існує велика кількість різноманітних ресурсів даних і те, як вони можуть змінити ставлення до них інвесторів, що беруть участь у ринку. Це призвело до зміщення алгоритмічних парадигм торгівлі, обсяг яких різко зростає з кожним роком. Цей тип торгівлі приваблює тих, хто прагне виграти на покупці, застосовуючи значну кількість різних методик. Однією з найбільш важливих цілей автоматизованої торгівлі є "win a market", тобто бути максимально точним при прогнозуванні майбутнього прибутку, щоб отримати потенційну винагороду.[2]

Найбільш зручним способом перегляду даних аналізу та моніторингу фондового ринку буде веб застосунок, оскільки є можливість переглядати у повному масштабі на комп'ютері та у швидкому доступі зі смартфону.

**Практичне значення одержаних результатів.** Розроблено алгоритм прогнозування цін акцій на біржовому ринку.

# 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

## 1.1 Опис предметного середовища

Частину ринку капіталу, де цінні папери випускаються, купуються та продаються називається фондовим ринком (ринок цінних паперів). [3]

Ринок з певним видом цінних паперів або товарів, які є об'єктами частих і великих угод, називається активним ринком.

Саме поняття фондового ринку являється абстрактним, яке служить для визначення наборів дій та механізмів, які роблять торгівлю цінними паперами можливою.

Організаційно розроблений, функціонуючий увесь час ринок, де торгують цінними паперами; акціонерне товариство, яке фокусує пропозицію та попит цінних паперів, впливає на створення їх валютного курсу та діє відповідно до законодавства, правил та статуту біржі, називається фондовою біржою.

Акції, державні боргові зобов'язання, облігації, державні боргові зобов'язання, а також похідні від них є основними видами цінних паперів, що торгуються на фондовій біржі. Лістингом називається процедура допуску їх до біржових котирувань. Воно здійснюється відповідно до певних вимог їх емітента. Активи – це цінності фонду, усі цінні папери, що пройшли процедуру лістингу та дозволені до продажу. Операції торгівлі проводиться або партіями або за видами, а договори укладаються без наявності цінних паперів.

Біржа не являється комерційною організацією і не спрямована на отримання прибутку. Її дохід формується за рахунок регулярних членських внесків, продажу акцій, пені, обмінних зборів. Усі доходи використовуються для покриття витрат на забезпечення та розвиток діяльності біржі.

Основні учасники біржового ринку:

– емітенти - корпорації, що випускають цінні папери;

- інвестори - ті, що купують ціні папери:
- інвестиційні установи:
  - інвестиційний фонд;
  - фінансовий брокер;
  - інвестиційна компанія;
  - інвестиційний консультант;
  - посередник.

У результаті подвійного акціону або ранжування заявок у межах ціни закупівлі та ціни продажу формується ціна ціних паперів на біржі. Новий аукціон потребує окремий контракт. Курсові ціни фіксуються кілька разів на день і опубліковуються щодня в пресі, а також публікується інформація про укладені договори . Кожен рік Всесвітньою федерацією фондових бірж та Федерацією бірж Європейського співтовариства видають інформаційні бюлетені фондових бірж.

За допомогою різних показників, таких як індекси, аналізується стан фондової біржі та економіки в цілому. Для кожної біржі існує своя система індексів. Найвідоміші: Nikkei Index - Токійська фондова біржа, індекс Dow Jones - Нью-Йоркська фондова біржа, Dax Index - Німеччина. [4]

### 1.1.1 Опис процесу діяльності

У самому початку користувач реєструється у системі та формує свій профіль шляхом вибору напрямків інвестування, визначенням цілей прибутку, термінів та рівня ризику. Система формує модель під користувача. Отримуючі дані з мережі про стан ринку, система формує рекомендації шляхом формування звіту. Користувач приймає рішення щодо продажу чи купівлі того чи іншого активу. Якщо користувач дозволяє системі замість нього приймати рішення, то це робить система. Замовлення відсилаються до брокера, який і проводить операцію.

Діаграма діяльності зображена на рисунку 1.1.

Рисунок 1.1 – Діаграма діяльності

### 1.1.2 Опис функціональної моделі

Актором системи є користувач. Користувач – формує напрями інвестування шляхом вибору інтересів, примає рішення щодо купівлі чи продажу активів. Діаграма варіантів використання зображена на рисунку 1.2.

Функції користувача:

– формування портфелю;

– прийнття рішення покупки чи продажу активів;

– введення чи виведення коштів.

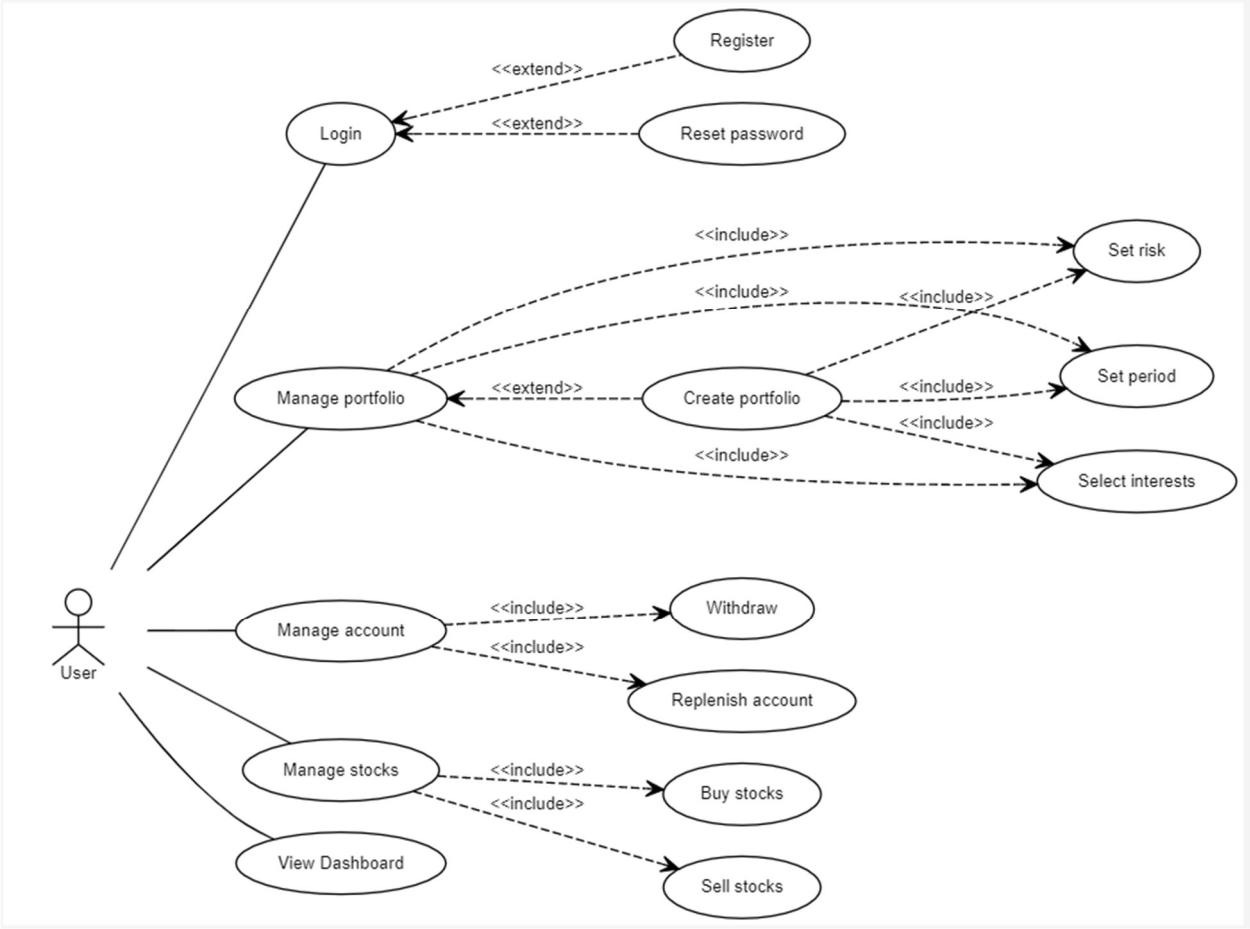



Рисунок 1.2 – Діаграма варіантів використання

## 1.2 Огляд наявних аналогів

Було виявлено кілька сервісів зі схожою функціональністю. Деякими з них є:

– https://8topuz.com/;

- https://www.hihedge.com/;
- https://ru.investing.com/technical/;

Розглянемо функціональності цих сервісів.

**https://8topuz.com**

Програмне забезпечення для торгівлі на валютному ринку

Даний алгоритм AI Trading Technology використовує запатентовану нейронну мережу, яка аналізує глибину ринку і шукає закономірності заданих математичних моделей (таких як фрактали, хаос і хвилі), що дозволяє виділяти і прогнозувати тенденції ринку в режимі реального часу.

У той час як машинне навчання може бути відносно новим терміном для фінансової індустрії, нейронні мережі добре відомі, особливо кванти.

Модель являє собою систему, орієнтовану на короткочасну волатильність і застосовує суворі принципи управління ризиками. Протягом мілісекунд система може вибрати найбільш підходящу стратегію з більш ніж 30 000 сценаріїв у всіх ринкових умовах.

**https://www.hihedge.com/**

hiHedge забезпечує торговельні стратегії, керовані AI, що перевищують людські можливості. На фінансових ринках є занадто багато точок даних, які люди не можуть інтерпретувати. Люди обмежені власним досвідом та наявними даними, що обмежує поточну алгоритмічну торгівлю, яку здійснює людина.

Він використовує технологію, схожу на AlphaGo Google DeepMind, використовуючи комбінацію алгоритмів глибокого навчання та посилення.

AI трейдер видобуває приховані тенденції, інформацію та взаємозв'язки за допомогою звивистих нейронних мереж, які можуть розпізнавати велику кількість наборів даних високих розмірів при розгляді даних мікро, макро та новин.

Завдяки глибокому вивченню підкріплення, трейдери з ІП можуть постійно вчитися та самостійно розробляти значні торгові рішення. **https://ru.investing.com/technical/**

Відображає технічний огляд по основних валютних парах, індексам, акціям, ф'ючерсах. Аналіз є цілковитою інформаційної зведенням даних по середнім ковзним, а також ключовим технічним індикаторам, які відображають певні періоди часу.

Є можливість створення технічного аналіз за власним списком фінансових інструментів в режимі реального часу. Вся інформація обновляється автоматично в реальному часі.

## 1.3 Постановка задачі

### 1.3.1 Призначення розробки

Інформаційна система моніторингу та аналізу біржових ринків призначена для ефективного спостереження за станом ринку та допомоги у прийнятті рішення щодо інвестування.

### 1.3.2 Цілі та задачі розробки

Метою даної інформаційної системи є підвищення прибутковості від інвестування шляхом пошуку менш ризикованих шляхів та спостереженням змін стану ринку, на основі яких створювати рекомендації щодо продажу чи купівлі цінних паперів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- ведення аккаунту користувача;
- створення портфелю користувача;

- збір даних щодо стану ринку;
- обробка даних з ринку;
- створення моделі, що відображає ринок;
- формування звітності.

**Висновок до розділу**

Виконано проєктування інформаційної системи моніторингу та аналізу біржових ринків:

- виконано аналіз предметного середовища дипломного проєкту;
- було розглянуто процес діяльності в інформаційній системі, відповідно до якого було виділено користувача, що взаємодіє з системою;
- визначено функції користувача у системі;
- розроблено порядок виконання дій акторів при роботі з інформаційною системою;
- було визначено призначення і цілі розробки та описані задачі, які необхідно реалізувати під час виконання дипломного проєкту.

# 2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

## 2.1 Вхідні дані

У таблиці 2.1 представлені вхідні дані.

Таблиця 2.1 – Вхідні дані

| *Данні* | *Опис* |
|---|---|
| Дані про портфель користувача | Дані користувача, призначені для опису опису портфелю користувача (напрямки інвестування, термін інвестування, рівень ризику) |
| Дані про стан ринку | Дані для опису стану ринку (наіменування ціного паперу, вартість, дата та час) |

## 2.2 Вихідні дані

У таблиці 2.2 представлені вихідні дані системи

Таблиця 2.2 – Вихідні дані

| *Дані* | *Опис* |
|---|---|
| Дані про прогнозування стану ринку | Прогнозована вартість акцій за певний період |
| Дані статистики стану ринку | Графік відображення зміни стану ринку за певний період |
| Дані статистики інвестування | Графіки інвестування з результатом за певний період |

## 2.3 Опис структури бази даних

У базі даних MS SQL сформовано таблиці, що зображені у таблиці 2.3:

Таблиця 2.3 – Таблиці бази даних

| **Назва таблиці** | **Назва стовпця** | **Тип даних** | **Детальна інформація** |
|---|---|---|---|
| Sector – таблиця секторів | Id | string | Ідентифікатор сектору |
| | Name | string | Назва сектору |
| | Description | string | Опис сектору |
| InvestTransaction – таблиця транзакцій користувача | Id | string | Ідентифікатор транзакції |
| | Amount | int | Кількість акцій |
| | StockId | string | Ідентифікатор акції |
| | OperationType | int | Тип операції |
| | Price | decimal | Проінвестована\виведена сума |
| | Time | datetime | Час транзакції |
| | CardAccountId | string | Ідентифікатор карткового рахунку |
| Portfolio – таблиця портфелю коритсувача | Id | string | Ідентифікатор |
| | UserId | string | Ідентифікатор користувача |
| | RiskId | string | Ідентифікатор ризику |
| | InvestPeriod | datetime | Бажаний період інвестування |
| | StartedTime | datetime | Початок інвестування |
| PortfolioStock – таблиця привязки зацікавлених користувачем | PortfolioId | string | Ідентифікатор портфоліо |
| | StockId | string | Ідентифікатор компаній |

| Назва таблиці | Назва стовпця | Тип даних | Детальна інформація |
|---|---|---|---|
| компаній до портфелю | | | |
| Risk – таблиця ризиків | Id | string | Ідентифікатор ризику |
| | Name | string | Назва ризику |
| | Description | string | Опис ризику |
| Stock – таблиця акцій компаній | Id | string | Ідентифікатор акцій |
| | MarketCap | long | Капіталізація компанії |
| | Symbol | string | Символ компанії на біржі |
| | TimeSeries | string | Історія цін у JSON форматі |
| | Name | string | Назва компанії |
| | SectorId | string | Ідентифікатор сектору до якого належить компанія |
| | Description | string | Опис компанії |
| CardAccount – таблиця карткових рахунків користувача | Id | string | Ідентифікатор рахунку |
| | PortfolioId | string | Ідентифікатор портфоліо до якого привязан рахунок |
| | Name | string | Назва рахунку |
| | IsDisabled | int | Прапор активності рахунку |
| | AddedDate | datetime | Час коли був додан рахунок |
| | CardNumber | string | Номер рахунку |
| | Id | string | Ідентифікатор ціни |

| Назва таблиці | Назва стовпця | Тип даних | Детальна інформація |
|---|---|---|---|
| StockPrice – таблиця цін акцій | StockId | string | Ідентифікатор акції |
| | OpeningPrice | decimal | Ціна на момент відкриття ринку |
| | ClosingPrice | decimal | Ціна на момент закриття ринку |
| | HighestPrice | decimal | Найвища ціна |
| | LowestPrice | decimal | Найнижча ціна |
| | Time | datetime | Дата |
| ApplicationUser – таблиця користувача системи | Id | string | Ідентифікатор користувача |
| | UserName | string | Імя користувача у системі |
| | LastName | string | Прізвище користувача |
| | FirstName | string | Імя користувача |
| | Email | string | Електронна пошта |
| | Picture | string | Картинку у форматі bas64 |
| | PasswordHash | string | Хеш пароля |
| MarketCap – таблиця обємів ринку | Id | String | Ідентифікатор обєму |
| | Name | String | Назва |
| | Description | String | Опис |
| | MaxCap | Long | Максимальне значення |
| | MinCap | long | Мінімальне значення |

Структурна схема бази даних представлена у графічному матеріалі.

**Висновок до розділу**

У даному розділі дипломного проєкту були сформовані вхідні дані інформаційної системи, та визначено Їх джерела. Також було описано вихідні дані системи та їх структура.

В даному розділі описані таблиці бази даних та кожне поле таблиці з описом даних, що зберігаються у ній.

Представлена структурна схема бази даних у графічному матеріалі, що показує зв’язки між таблицями.

# 3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

## 3.1 Змістовна постановка задачі

Інформаційна система містить функцію прогнозування динаміки індексів цін на фондовому ринку для ефективного прийняття рішень щодо інвестування коштів у певну галузь та зменшення інвестиційного ризику. Для досягнення поставленої мети необхідно виконати такі завдання: визначення факторів, що впливають на зміну індексів цін; розробка алгоритму навчання; визначення нейронної мережі; визначення ефективності моделі та перевірка її точності.

## 3.2 Математична постановка задачі

Нехай часовий ряд в дискретні моменти часу t = 1,2, ..., T - цінові індекси. Значення історичних цінових індексів надані у такому вигляді W (t) = W (1), W (2), ..., W (T). У момент часу T необхідно визначити значення процесу W (T) в моменти часу T + 1, T + 2, ..., T + P. Момент часу T - момент прогнозу, а величина P називається часом попередження, часова мітка, на яку потрібно зробити прогноз.

Для обчислення значень часового ряду в майбутні моменти часу потрібно визначити функціональну залежність, яка відображатиме зв'язок між минулими і майбутніми значеннями цього ряду

$$W(t) = F(W(t-1), W(t-2), W(t-3), \dots) + \varepsilon_t \quad (1)$$

Ця залежність (1) являє собою модель прогнозування. Необхідно створити таку модель прогнозування, для якої абсолютне середнє відхилення правильного значення від прогнозованого спрямовується до мінімального для заданого P.

### 3.3 Обґрунтування методу розв’язання

Для прогнозування була обрана архітектура рекурентної нейронної мережі (RNN), побудованої на елементах довгої короткострокової пам'яті (LSTM). [5] Оскільки історія руху цін є часовий ряд, то нам необхідно регулярно звертатися до неї і враховувати довгостроковий контекст для чого, власне, RNN з LSTM і призначені. Моделі Long Short-Term Memory - надзвичайно потужні моделі часових рядів. Вони можуть передбачити довільну кількість кроків у майбутнє. [6]

### 3.4 Опис методів розв’язання

Модуль LSTM (або комірка) містить 5 основних компонентів, що дозволяє моделювати як довгострокові, так і короткострокові дані. Модуль наведений на рисунку 3.1.

Стан комірки (c(t)) - визначає внутрішню пам'ять комірки, яка зберігає як короткочасну пам'ять, так і довгострокову пам'ять.

Прихований стан (h(t)) - вихідна інформація про стан, обчислена w.r.t. поточний вхід, попередній прихований стан та поточний вхід клітини, який використовується для прогнозування майбутніх цін на фондовому ринку. Крім того, прихований стан може вирішити відновити короткострокову чи довгострокову пам’ять або обидва типи пам'яті, що зберігаються в стані комірки, щоб зробити наступне передбачення.

Вхідні ворота (i(t)) - визначає, скільки інформації з поточного вводу надходить у стан комірки.

Забуваючі ворота (f(t)) - визначає, скільки інформації з поточного вводу та попереднього стану комірки перетікає у поточний стан комірки.

Вихідні ворота (o(t)) - визначають, скільки інформації з поточного стану комірки перетікає у прихований стан, так що при необхідності LSTM може вибирати лише довгострокові спогади або короткострокові та довгострокові спогади.

Рисунок 3.1 - Комірка LSTM моделі.

Рівняння для обчислення кожної з цих сутностей наступні:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$

$$\tilde{c}_t = \sigma(W_c[h_{t-1}, x_t] + b_c) \quad (3)$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (4)$$

$$c_t = f_t * c_t + i_t * \tilde{c}_t \quad (5)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = o_t * tanh(c_t) \quad (7)$$

Алгоритм тренування моделі LSTM:

1. Визначити тестовий набір вихідних точок у часовому ряду для оцінки моделі
2. Для кожної епохи:

   2.1.Для повної тривалості послідовності даних про навчання

   2.1.1. Розгорнути набір партій

   2.1.2. Тренувати нейронну мережу з розгорнутими партіями

   2.2.Обчислити середню втрату тренувань

2.3. Для кожної вихідної точки тестового набору

2.3.1. Оновити стан LSTM шляхом ітерації по попередніх точок, знайдених перед тестовою точкою

2.3.2. Зробити прогнози на задану кількість кроків, використовуючи попереднє прогнозування як поточний вхід

2.3.3. Обчислити втрати MSE між прогнозованими точками та справжніми цінами на акції в ці часові точки

Для кожної часової точки отримаємо набір прогнозованих значень зміни цін, візулізацію яких можна побачити на рисунку 3.2.

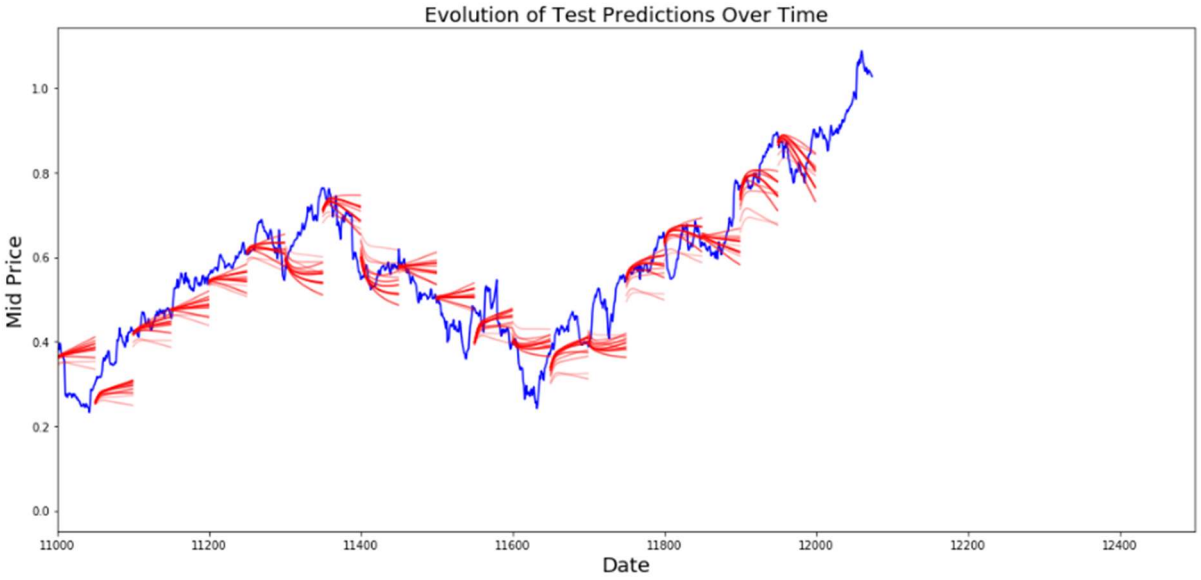



Рисунок 3.2 - Приклад прогнозованих значень для часових точок з різною кількість епох тренування.

Вибираємо епоху з найменшою похибкою та зберігаємо цю модель для подальшого використання. На рисунку 3.3 зображено результат для кращої епохи.

Рисунок 3.3 - Приклад прогнозованих значень для часових точок для епохи тренування з кращою результативністю.

**Висновок до розділу**

У розділі було розглянуто математичну та змістовну постановку задачі. Також був представлений та описаний алгоритм розв’язку поставленої задачі.

# 4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

## 4.1 Засоби розробки

Розробку програмного забезпечення для інформаційної системи було розділено на дві частини: клієнтську та серверну.

Для розробки серверної частини було використано ASP.NET Core - це крос-платформа, високоефективна система з відкритим вихідним кодом для створення сучасних додатків, підключених до хмар, підключених до Інтернету.[7]

Мільйони розробників використовують або використовували ASP.NET 4.x для створення веб-додатків. ASP.NET Core - це редизайн ASP.NET 4.x, включаючи архітектурні зміни.

ASP.NET Core надає такі переваги:

- Уніфікована історія для створення веб-інтерфейсу та веб-API.
- Архітектурно встановлений для встановлення.
- Сторінки Razor Pages полегшують та продуктивніші кодування сценаріїв, орієнтованих на сторінку.
- Blazor дозволяє використовувати C # у браузері поряд з JavaScript. Діліться логікою додатків на стороні сервера та клієнта всім написаним на .NET.
- Можливість розробки та роботи на Windows, macOS та Linux.
- Відкритий і орієнтований на громаду
- Інтеграція сучасних структур на базі клієнта та робочих процесів розвитку.
- Підтримка хостингу послуг віддаленого виклику процедур (RPC) за допомогою gRPC.
- Система конфігурації, орієнтована на хмару.
- Вбудована ін'єкційна залежність.

– Легкий, високопродуктивний та модульний протокол HTTP-запитів.
– Інструменти, що спрощують сучасну веб-розробку.

СУБД MsSQL використовується для управління базою даних.

Для роботи з базою даних було використано Entity Framework Core. Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і расширяемую технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але має більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

Alpha Vantage API було використано для використання історичних дан их. Цей сервіс може надавати історичні дані по акціям за певний період.

Для розробки клієнтської частини було використано Angular 9, HTML, CSS, Bootstrap. [8]

Angular - це структура дизайну прикладних програм та платформа розробки для створення ефективних та складних односторінкових додатків.

HTML - це стандартна мова розмітки для створення веб-сторінок.

Каскадні таблиці стилів (CSS) - це мова таблиці стилів, яка використовується для опису подання документа, написаного в HTML або XML (включаючи діалекти XML, такі як SVG, MathML або XHTML). CSS описує, як елементи мають бути відображені на екрані, на папері, у мовленні або на інших носіях інформації.

## 4.2 Вимоги до технічного забезпечення

### 4.2.1 Загальні вимоги

Параметри системи, що необхідні для роботи серверу:

– RAM об’ємом 2 Гб

– CPU з частотою 2.4 ГГц ;

– HDD - мінімум 20 Гб

Необхідно, щоб на сервері було встановлене наступне програмне забезпечення:

– Windows

– MSSQL

– ASP.NET Core Runtime 3.1

Однією вимогою до клієнту – це наявність одного з веб-браузерів, таких як Edge, Google Chrome, Safari

## 4.3 Архітектура програмного забезпечення

### 4.3.1 Діаграма класів

Структурна схема класів представлена у графічному матеріалі, класи відповідають за виконання таких функцій, як представлення моделі даних.

Діаграма містить тринадцять класів, а саме:

- «StockPrice» - відповідає за представлення моделі даних цін акцій компаній;
- «Stock» - відповідає за представлення моделі даних компаній;
- «PortfolioStock» - відповідає за представлення моделі даних звязки портфоліо та компанії;
- «PredictionModel» - відповідає за представлення моделі даних прогнозу;
- «MarketCap» - відповідає за представлення моделі даних об’єму ринку компаній;

– «Portfolio» - відповідає за представлення моделі даних портфоліо користувача;

– «ApplicationUser» - відповідає за представлення моделі даних користувача системи;

– «Sector» - відповідає за представлення моделі даних сектору ринку до якого відноситься компанія;

– «PortfolioSector» - відповідає за представлення моделі даних звязки портфоліо користувача та сектору ринку;

– «OperationType» - відповідає за представлення моделі даних типу операції при транзакції;

– «InvestTransaction» - відповідає за представлення моделі даних транзакції;

– «CardAccount» - відповідає за представлення моделі даних рахунку користувача;

– «Risk» - відповідає за представлення моделі даних ризику;

### 4.3.2 Діаграма послідовності

У таблиці 4.1 представлені об’єкти що задяні на діаграмі послідовності.

Таблиця 4.1 – Об’єкти діаграми послідовності

| *Об’єкт* | *Відповідальність* |
|---|---|
| User | Введення даних авторизації, вибір компаній, прийняття рішення щодо купівлі\продажу акцій |
| Client | Відображає сайт, з яким повинен взаємодіяти користувач. |
| Server | Авторизує користувача, віддає список компаній, прогнозує рух вартості акцій |
| Stock market api | Повертає історичні ціни акцій |

Діаграма послідовності зображена на структурній схемі послідовності у графічному матеріалі.

### 4.3.3 Діаграма розгортування

Діаграма розгортування зображена на рисунку 4.1.

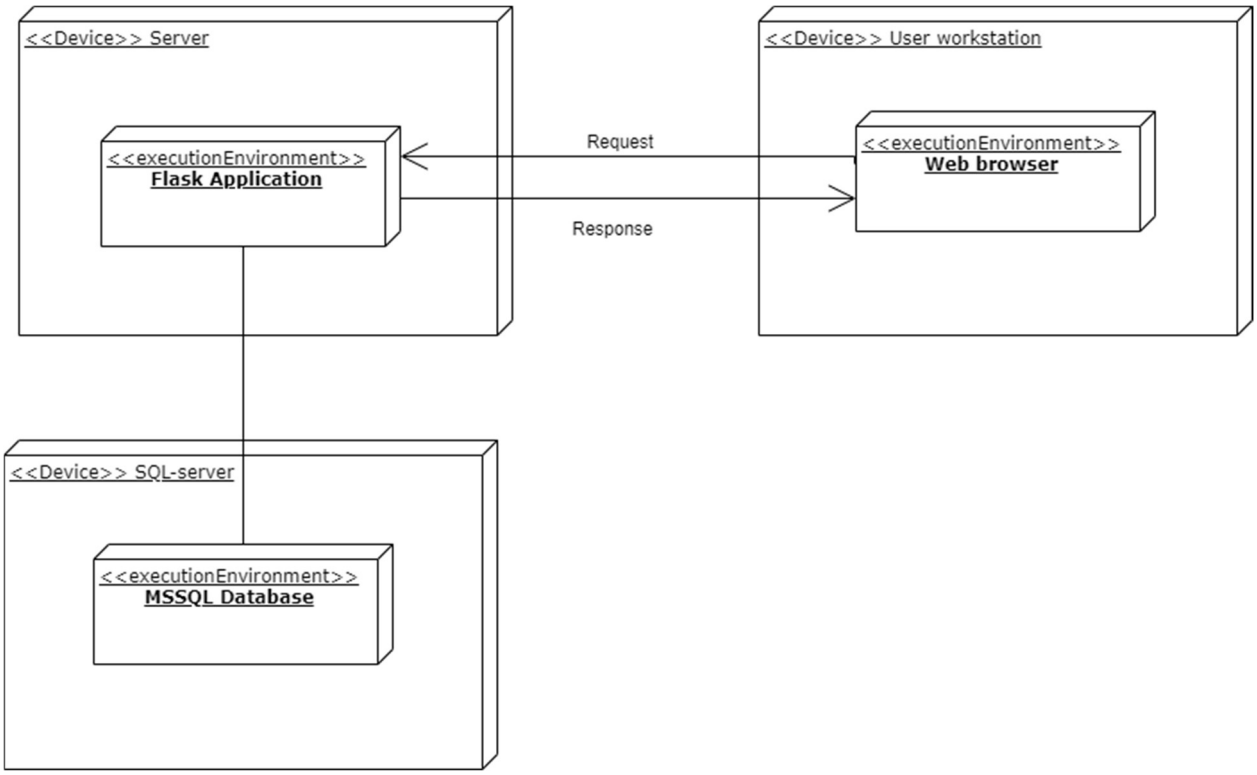



Рисунок 4.1 – Діаграмма розгортування

### 4.3.4 Специфікація функцій

У таблиці 4.2 наведені функції класів програмного забезпечення.

Таблиця 4.2 – Функції класів програмного забезпечення

| Назва | Примітка |
|---|---|
| **CardService** – клас, що відповідає за сервіс управління рахунками користувача | |
| Task<CardAccount[]> GetCardAccounts(string userId); | Повертає список рахунків користувача |

| Назва | Примітка |
|---|---|
| Task<CardAccount> CreateCardAccount(string userId, CardViewModel cardViewModel); | Створює рахунок користувача |
| Task RemoveCardAccount(string userId, string cardAccountId); | Видаляє рахунок користувача |
| **Клас: CompaniesService** – клас, що відповідає за сервіс управління компаніями | |
| Task<List<Stock>> GetStocks(List<string> sectors, List<string> marketCapacities, Portfolio portfolio); | Повертає список компаній |
| Task<List<Sector>> GetSectors(); | Повертає сектори ринку |
| Task<List<MarketCap>> GetMarketCapacities(); | Повертає об’єми ринку |
| **Клас: PortfolioService** – клас, що відповідає за сервіс управління портфоліо користувача | |
| Task<Portfolio> GetByUserId(string userId); | Повертає портфоліо користувача |
| Task<PortfolioStock> AddCompany(string userId, string stockId); | Додає компанію до портфоліо користувача |
| Task DeleteCompany(string userId, string stockId); | Видаляє компанію з портфоліо користувача |

| Назва | Примітка |
|---|---|
| Task<InvestTransaction> MakeOperation(OperationType operationType, OperationViewModel operationViewModel, string userId); | Проводить транзакцію купівлі\продажу акцій |
| Task<InvestTransaction[]> GetTransactions(string userId); | Повертає транзакції користувача |
| Task<decimal> GetEarnings(string userId); | Повертає зароблену суму коштів за весь період |
| **Клас: PredictionService** – клас, що відповідає за сервіс прогнозування цін акцій компаній | |
| public Task<PredictionModel> GetPrediction(string stockId, string userId); | Прогнозує поведінку зміни ціни акції |
| **Клас: StockService** – клас, що відповідає за сервіс отримання інформації щодо акцій компаній | |
| Task<StockTimeSeries> GetTimeSeries(string stockId); | Повертає історичні ціни акцій по дням |
| Task<decimal> GetCurrentPrice(string stockId); | Повертає ціну акції на даний момент |
| Task<int> GetAmount(string stockId, string userId); | Повертає кількість куплених акцій користувачем |
| **Клас: UserProfileService** – клас, що відповідає за сервіс управління профілем користувача | |

| Назва | Примітка |
|---|---|
| Task<UserProfile> GetUserProfileById(string id); | Повертає профіль користувача |
| Task<UserProfile> UpdateUserProfile(string userId, UserProfile userProfile); | Оновлює профіль користувача |
| Task<UserProfile> UpdateUserPicture(string userId, string picture); | Оновлює зображення користувача |
| **Клас: UserService** – клас, що відповідає за сервіс авторизації користувача в інформаційній системі | |
| Task<TokenViewModel> Register(RegisterViewModel registerViewModel); | Реєструє нового користувача у системі |
| Task<TokenViewModel> Login(LoginViewModel loginViewModel); | Авторизує користувача у системі |
| Task Reset(ResetViewModel resetViewModel); | Скидає пароль користувача |
| Task Logout(); | Виконує вихід користувача з системи |

### 4.4 Опис звітів

На рисунку 4.2 зображено приклад звіту системи по компанії.

Рисунок 4.2 - Звіт по компанії Майкрософт

Звіт містить інформацію по історичній ціні акції за останнє півріччя. Історичні ціни представленні за допомогою графіка японських свічок.

Свічковий графік відображає цілий ряд значень інформації за ціною, таких як ціна відкриття, ціна закриття, максимальна і мінімальна ціна, і використовує для цього графічні символи в формі свічок, кожна з яких дає короткий огляд результатів торговельної діяльності за певний період часу (хвилину, годину , день, місяць і так далі). Кожна свічка розташовується уздовж часової шкали на осі X і відображає підсумки торгової активності за певний період часу. [9]

Також звіт містить інформацію щодо кількості куплених користувачем акцій, ціну акції на даний момент часу та прогноз щодо купівлі.

**Висновок до розділу**

У цьому розділі дипломного проєкту детально описані засоби розробки інформаційної системи. Також у цьому розділі описані вимоги до технічного та програмного забезпечення, які потрібні для даної інформаційної системи. Дані вимоги до технічного забезпечення включають вимоги клієнтської частини та серверної частини

У розділі наведено діаграми послідовності та компонентів, які описують архітектуру системи та її частин,  класів, також описані функції класів.

Описані звіти інформаційної системи.

# 5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

## 5.1 Керівництво користувача

Для запуску веб-додатку треба відкрити у будь-якому браузері адресу веб-додатку. При першому відкритті додатку буде відображена форма входу до веб додатку (рисунок 5.1). Для авторизації потрібно ввести пошту та пароль.




Рисунок 5.1 – Авторизація користувача

Якщо користувача немає в системі, то потрібно перейти на сторінку реєстрації (рисунок 5.2). При реєстрації потрібно ввести пошту, ім’я користувача в системі та пароль.




Рисунок 5.2 – Реєстрація користувача

Після успішної авторизації, заходимо у профіль користувача, зображення сторінки профілю показано на рисунку 5.3. Тут є можливість змінити ім’я, пошту, та зображення аватарки користувача.

Рисунок 5.3 – Профіль користувача

На сторінці портфелю користувача є можливість переглянути список компаній за якими слідкує користувач. Сторінка портфелю зображена на рисунку 5.4.




Рисунок 5.4 – Портфоліо користувача

Для довання компанії потрібно натиснути на кнопку «Add company». На сторінці додавання компанії є можливість пошуку компанії по заданим параметрам, таких, як: назва, тип сектору на ринку, капіталізація компанії. Компанія додається натисканням кнопки «Add» у відповідній компанії. Сторінка додавання компанії відображається на рисунку 5.5.




Рисунок 5.5 – Додавання компанії до портфоліо користувача

Для перегляду звітності по компаніям потрібно перейти на сторінку «Dashboard» (рисунок 5.6). На цій сторінці відображється звітність по обраним компаніям. Деталі по звітності розписані у пункті 4.4.

Рисунок 5.6 – Дашборд

Для купівлі акцій натискаємо кнопку «Buy». З'являється попап із покупкою акцій, зображений на рисунку 5.7. Для купівлі акції потрібно ввести бажану кількість акцій та обрати рахунок користувача.




Рисунок 5.7 – Купівля акцій

Для продажу акцій натискаємо кнопку «Sell». З’являється попап із продажем акцій, зображений на рисунку 5.8. Для продажу акції потрібно ввести бажану кількість акцій та обрати рахунок користувача.




Рисунок 5.8 – Продаж акцій

Для перегляду історій купівель\продаж акцій потрібно перейти на сторінку «Account» (рисунок 5.9). На цій сторінці зображено історія усіх транзакцій та загальний прибуток за весь період.

Рисунок 5.9 – Транзакції користувача

Для додавання рахунку потрібно натиснути на кнопку «Add card». Сторінка за додаванням рахунку зображена на рисунку 5.10. Для додавання рахнку потрібно ввести ім’я рахунку та номер картки.




Рисунок 5.10 – Додавання рахунку

## 5.2 Випробування програмного продукту

### 5.2.1 Мета випробувань

Метою випробувань є перевірка відповідності функцій інформаційної системи моніторингу та аналізу біржових ринків вимогам технічного завдання.

### 5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

### 5.2.3 Результати випробувань

У процесі тестування програмного забезпечення було перевірено уся функціональність інформаційної системи (ІС). У таблицях 5.1 – 5.7 наведений перелік випробувань основних функціональних можливостей.

Таблиця 5.1 – Реєстрація користувача

| **Мета тесту:** | **Перевірка функції «Реєстрація користувача»** |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | Username, email, password. |
| Схема проведення тесту: | Перейти на вкладку «реєстрація» |
| Очікуваний результат: | Користувач додається до системи |
| Стан ІС після проведення випробувань: | Система містить дані по новому користувачу |

Таблиця 5.2 – Авторизація користувача

| **Мета тесту:** | **Перевірка функції «Авторизація користувача»** |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | email, password. |

| **Мета тесту:** | **Перевірка функції «Авторизація користувача»** |
|---|---|
| Схема проведення тесту: | Перейти на вкладку «авторизація» |
| Очікуваний результат: | Користувач успішно авторизується у системі |
| Стан ІС після проведення випробувань: | Користувач авторизований у системі |

Таблиця 5.3 – Отримання списку компаній

| **Мета тесту:** | **Перевірка функції «Отримання списку компаній»** |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | Назва, тип сектору, об’єм ринку |
| Схема проведення тесту: | Перейти на вкладку «портфоліо», додавання компанії |
| Очікуваний результат: | Повертається список компаній по заданим фільтрам |
| Стан ІС після проведення випробувань: | Відображається список компаній по заданим фільтрам |

Таблиця 5.4 – Отримання звітності

| **Мета тесту:** | **Перевірка функції «Отримання звітності»** |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | |
| Схема проведення тесту: | Перейти на вкладку «дашборду» |
| Очікуваний результат: | Повертається звіт по компанії |

| Мета тесту: | Перевірка функції «Отримання звітності» |
|---|---|
| Стан ІС після проведення випробувань: | Відображається звіт по компанії |

Таблиця 5.5 – Купівля акцій

| Мета тесту: | Перевірка функції «Купівля акцій» |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | Кількість акцій, рахунок користувача, назва компанії |
| Схема проведення тесту: | Перейти на вкладку «дашборду», купівля |
| Очікуваний результат: | У системі створюється транзакція купівлі акцій |
| Стан ІС після проведення випробувань: | Система містить транзакцію купівлі акцій |

Таблиця 5.6 – Продаж акцій

| Мета тесту: | Перевірка функції «Продаж акцій» |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | Кількість акцій, рахунок користувача, назва компанії |
| Схема проведення тесту: | Перейти на вкладку «дашборду», продаж |
| Очікуваний результат: | У системі створюється транзакція продажу акцій |
| Стан ІС після проведення випробувань: | Система містить транзакцію продажу акцій |

Таблиця 5.7 – Отримання списку транзакцій

| **Мета тесту:** | **Перевірка функції «Отримання списку транзакцій»** |
|---|---|
| Початковий стан ІС | Сервер запущений та працює. |
| Вхідні данні: | |
| Схема проведення тесту: | Перейти на вкладку «рахунки» |
| Очікуваний результат: | Повертається список транзакцій по користувачу |
| Стан ІС після проведення випробувань: | Відображається список транзакцій по користувачу |

**Висновок до розділу**

У даному розділі дипломного проєкту наведено керівництво користувача, яке включає покроковий опис дій які можуть бути виконані у системі, з рисунками веб сторінок для показу результатів виконання дій користувачем. Керівництво користувача містить опис дій які виконуються у веб-додатку, а саме: реєстрації, авторизації, управлінням портфоліо, управлінням профілем, управління рахунками, купівлею та продажем акцій.

Представлені сценарії для тестування інформаційної системи щодо відповідності функціональним вимогам.

## ЗАГАЛЬНІ ВИСНОВКИ

Під час реалізації дипломного проєкту було досліджено предметну область фондових ринків, визначено вимоги до інформаційної системи. Визначаються вхідні дані вхідної інформаційної системи та їх джерела. Визначаються вихідні дані інформаційної системи, наводиться їх структура. Описані та зображені процеси та функції реалізуються в інформаційній системі за допомогою діаграм діяльності та випадків використання.

На основі даних, отриманих під час аналізу, було сформульовано завдання створити модель прогнозування на основі історичних даних. Вибирається модель, на основі якої обчислюється прогноз ціни акцій.

Для розробки програмного забезпечення для інформаційної системи були використані такі інструменти: мова C # та Asp .Net Core для серверного сервісу, мова шрифту та кутова рамка веб-програми. Наведено опис засобів розробки та перелік основних переваг вибраних інструментів.

Розроблена модель бази даних, яка дозволяє зберігати та отримувати доступ до даних інформаційної системи. MS SQL використовується для управління базою даних.

Інструкція з експлуатації інформаційної системи для веб-програми. Описано кроки, необхідні для виконання всіх функцій інформаційної системи; надаються копії веб-сторінок. Описані тести програмного продукту на відповідність функціональним вимогам.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Stock and bonds [Електронний ресурс] – Режим доступу до ресурсу: https://www.khanacademy.org/economics-finance-domain/core-finance/stock-and-bonds
2. Algorithmic Trading 101 – Lesson1:Time Series Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/the-ocean-trade/the-ocean-x-algo-trading-lesson-1-time-series-analysis-fa3b76f1d4a3
3. Основы фондового рынка [Електронний ресурс] – Режим доступу до ресурсу: https://traders-union.ru/osnovi_fondovogo_rinka/.
4. Машинное обучение: что нужно знать о создании стратегий для торговли на бирже. Часть IV [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/iticapital/blog/281603/.
5. Эксперимент: создание алгоритма для прогнозирования поведения фондовых индексов [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/en/company/iticapital/blog/278023/
6. Understanding LSTM Networks [Електронний ресурс] – Режим доступу до ресурсу: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
7. Asp .net documentation [Електронний ресурс] – Режим доступу до ресурсу https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1.
8. Angular docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs.
9. Г. Моррис Японские свечи. Метод анализа акций и фьючерсов, проверенный временем. – 2012. – 312 с.

**Додаток А**

## *Тексти програмного коду*

***Інформаційна система моніторингу та аналізу біржових ринків***

(Найменування програми (документа))

*DVD-R*

(Вид носія даних)

*50*

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020 року

```
using System;
using System.Threading;

public partial class Page : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      Response.ContentType = "text/html";

      int lastIndex = Request.PathInfo.LastIndexOf("/");
      string pageNumber = (lastIndex == -1 ? "Unknown" : Request.PathInfo.Substring(lastIndex +
1));
      if (!string.IsNullOrEmpty(Request.QueryString["pageNumber"]))
      {
         pageNumber = Request.QueryString["pageNumber"];
      }
      Response.Output.Write("<html><head><title>Page" + pageNumber + "</title></head>");
      Response.Output.Write("<body>Page number <span id=\"pageNumber\">");
      Response.Output.Write(pageNumber);
      //Response.Output.Write("<script>var s=''; for (var i in window) {s += i + ' -> ' + window[i] +
'<p>';} document.write(s);</script>")'
      Response.Output.Write("</span></body></html>");
   }
}
using System;

public partial class Redirect : Page
{
   protected new void Page_Load(object sender, EventArgs e)
   {
      Response.Redirect("resultPage.html");
   }
}
using System.Threading.Tasks;
using AlgoMe.Services;
using AlgoMe.Services.Interfaces;
using AlgoMe.ViewModel;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   public class AuthController : Controller
```

```
  {
    private readonly IUserService _userService;

    public AuthController(IUserService userService)
    {
      _userService = userService;
    }

    [HttpPost("login")]
    public async Task<TokenViewModel> Login([FromBody] LoginViewModel loginViewModel)
    {
      return await _userService.Login(loginViewModel);
    }

    [HttpPost("register")]
    public async Task<TokenViewModel> Register([FromBody] RegisterViewModel
registerViewModel)
    {
      return await _userService.Register(registerViewModel);
    }

    [HttpPost("request-pass")]
    public async Task<IActionResult> RequestPass([FromBody] ResetViewModel resetViewModel)
    {
      return Ok();
    }

    [HttpPut("reset-pass")]
    public async Task<IActionResult> ResetPass([FromBody] ResetViewModel resetViewModel)
    {
      return Ok();
    }

    [HttpDelete("logout")]
    public async Task<IActionResult> Logout()
    {
      return Ok();
    }
  }
}
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Services;
using AlgoMe.Services.Interfaces;
using AlgoMe.ViewModel;
```

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
   [ApiController]
   [Authorize]
   [Route("api/[controller]")]
   public class CardController : Controller
   {
      private readonly ICardService _cardService;
      private readonly SignInManager<ApplicationUser> _signInManager;

      public CardController(ICardService cardService, SignInManager<ApplicationUser>
signInManager)
      {
         _cardService = cardService;
         _signInManager = signInManager;
      }

      [HttpGet(nameof(GetCardAccounts))]
      public async Task<IActionResult> GetCardAccounts()
      {
         var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);

         var cardAccounts = await _cardService.GetCardAccounts(userId);

         return Ok(cardAccounts);
      }

      [HttpPost(nameof(CreateCardAccount))]
      public async Task<IActionResult> CreateCardAccount([FromBody] CardViewModel
cardViewModel)
      {
         var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);

         var cardAccount = await _cardService.CreateCardAccount(userId, cardViewModel);

         return Ok(cardAccount);
      }

      [HttpDelete(nameof(DeleteCardAccount))]
      public async Task<IActionResult> DeleteCardAccount([FromQuery] string cardAccountId)
      {
```

```
            var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
            await _cardService.RemoveCardAccount(userId, cardAccountId);
            return Ok();
        }
    }
}
using System.Collections.Generic;
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Services.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
    [ApiController]
    [Authorize]
    [Route("api/[controller]")]
    public class CompaniesController : Controller
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ICompaniesService _companiesService;
        private readonly IPortfolioService _portfolioService;

        public CompaniesController(ICompaniesService companiesService,
            SignInManager<ApplicationUser> signInManager,
            IPortfolioService portfolioService)
        {
            _companiesService = companiesService;
            _signInManager = signInManager;
            _portfolioService = portfolioService;
        }

        [HttpGet("[action]")]
        public async Task<IActionResult> GetStocks([FromQuery]List<string> sectors,
            [FromQuery]List<string> marketCapacities,
            [FromQuery]bool filterUsed)
        {
            Portfolio portfolio = null;
            if (filterUsed)
            {
                var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
                portfolio = await _portfolioService.GetByUserId(userId);
            }
```

```
            var companies = await _companiesService.GetStocks(sectors, marketCapacities, portfolio);
            return Ok(companies);
        }

        [HttpGet("[action]")]
        public async Task<IActionResult> GetSectors()
        {
            var sectors = await _companiesService.GetSectors();
            return Ok(sectors);
        }

        [HttpGet("[action]")]
        public async Task<IActionResult> GetCapacities()
        {
            var marketCapacities = await _companiesService.GetMarketCapacities();
            return Ok(marketCapacities);
        }
    }
}
using System.Threading.Tasks;
using AlgoMe.Temp;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class InfoController : Controller
    {
        private readonly GenerateDbService _generateDbService;

        public InfoController(GenerateDbService generateDbService)
        {
            _generateDbService = generateDbService;
        }

        [HttpGet]
        public async Task<IActionResult> Get()
        {
            await _generateDbService.ProcessFile();
            return Ok();
        }
    }
}
using Microsoft.AspNetCore.ApiAuthorization.IdentityServer;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace AlgoMe.Controllers
{
   public class OidcConfigurationController : Controller
   {
      private readonly ILogger<OidcConfigurationController> _logger;

      public OidcConfigurationController(IClientRequestParametersProvider
clientRequestParametersProvider,
         ILogger<OidcConfigurationController> logger)
      {
         ClientRequestParametersProvider = clientRequestParametersProvider;
         _logger = logger;
      }

      public IClientRequestParametersProvider ClientRequestParametersProvider { get; }

      [HttpGet("_configuration/{clientId}")]
      public IActionResult GetClientRequestParameters([FromRoute] string clientId)
      {
         var parameters = ClientRequestParametersProvider.GetClientParameters(HttpContext,
clientId);
         return Ok(parameters);
      }
   }
}
using System;
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Services.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;

namespace AlgoMe.Controllers
{
   [ApiController]
   [Authorize]
   [Route("api/[controller]")]
   public class PortfolioController : Controller
   {
      private readonly IPortfolioService _portfolioService;
```

```
private readonly SignInManager<ApplicationUser> _signInManager;

public PortfolioController(IPortfolioService portfolioService,
   SignInManager<ApplicationUser> signInManager)
{
   _portfolioService = portfolioService;
   _signInManager = signInManager;
}

[HttpGet]
public async Task<IActionResult> Get()
{
   var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
   var portfolio = await _portfolioService.GetByUserId(userId);
   return Ok(portfolio);
}

[HttpGet(nameof(AddCompany))]
public async Task<IActionResult> AddCompany([FromQuery] string stockId)
{
   var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
   var portfolioStock = await _portfolioService.AddCompany(userId, stockId);
   return Ok(portfolioStock);
}

[HttpGet(nameof(DeleteCompany))]
public async Task<IActionResult> DeleteCompany([FromQuery] string stockId)
{
   var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
   await _portfolioService.DeleteCompany(userId, stockId);
   return Ok();
}

[HttpPost(nameof(MakeOperation))]
public async Task<IActionResult> MakeOperation([FromQuery] int operationType,
   [FromBody] OperationViewModel operationViewModel)
{
   var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
   var ot = Math.Sign(operationType) == 1 ? OperationType.Sell : OperationType.Buy;
   var result = await _portfolioService.MakeOperation(ot, operationViewModel, userId);
   return Ok(result);
}

[HttpGet(nameof(GetTransactions))]
public async Task<IActionResult> GetTransactions()
```

```
        {
            var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
            var result = await _portfolioService.GetTransactions(userId);
            return Ok(result);
        }

        [HttpGet(nameof(GetEarnings))]
        public async Task<IActionResult> GetEarnings()
        {
            var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
            var result = await _portfolioService.GetEarnings(userId);
            return Ok(result);
        }
    }

    public class OperationViewModel
    {
        [JsonProperty("stockId")]
        public string StockId { get; set; }
        [JsonProperty("accountId")]
        public string AccountId { get; set; }
        [JsonProperty("amount")]
        public int Amount { get; set; }
    }
}
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Services;
using AlgoMe.Services.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
    [ApiController]
    [Authorize]
    [Route("api/[controller]")]
    public class StockController : Controller
    {
        private readonly IStockService _stockService;
        private readonly IPredictionService _predictionService;
        private readonly SignInManager<ApplicationUser> _signInManager;

        public StockController(IStockService stockService,
```

```
        IPredictionService predictionService,
        SignInManager<ApplicationUser> signInManager)
    {
        _stockService = stockService;
        _predictionService = predictionService;
        _signInManager = signInManager;
    }

    [HttpGet(nameof(GetDailyTimeSeries))]
    public async Task<IActionResult> GetDailyTimeSeries([FromQuery] string stockId)
    {
        var result = await _stockService.GetTimeSeries(stockId);
        return Ok(result);
    }

    [HttpGet(nameof(GetStockPrediction))]
    public async Task<IActionResult> GetStockPrediction([FromQuery] string stockId)
    {
        var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
        var prediction = await _predictionService.GetPrediction(stockId, userId);
        return Ok(prediction);
    }

    [HttpGet(nameof(GetCurrentPrice))]
    public async Task<IActionResult> GetCurrentPrice([FromQuery] string stockId)
    {
        var price = await _stockService.GetCurrentPrice(stockId);
        return Ok(price);
    }

    [HttpGet(nameof(GetAmount))]
    public async Task<IActionResult> GetAmount([FromQuery] string stockId)
    {
        var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
        var amount = await _stockService.GetAmount(stockId, userId);
        return Ok(amount);
    }
  }
}
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Services;
using AlgoMe.Services.Interfaces;
using AlgoMe.ViewModel;
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace AlgoMe.Controllers
{
   [ApiController]
   [Authorize]
   [Route("api/[controller]")]
   public class UserProfileController : Controller
   {
      private readonly SignInManager<ApplicationUser> _signInManager;
      private readonly IUserProfileService _userService;

      public UserProfileController(IUserProfileService userService, SignInManager<ApplicationUser>
signInManager)
      {
         _userService = userService;
         _signInManager = signInManager;
      }

      [HttpGet]
      public async Task<UserProfile> Get()
      {
         var user = await
_signInManager.UserManager.GetUserAsync(_signInManager.Context.User);

         return new UserProfile
         {
            Email = user.Email,
            FirstName = user.FirstName ?? "",
            Id = user.Id,
            LastName = user.LastName ?? "",
            Picture = user.Picture ?? ""
         };
      }

      [HttpGet("picture")]
      public async Task<IActionResult> GetPicture()
      {
         var user = await
_signInManager.UserManager.GetUserAsync(_signInManager.Context.User);

         return Json(new { picture = user.Picture });
      }
```

```
        [HttpPut]
        public async Task<UserProfile> Put([FromBody] UserProfile userProfile)
        {
            var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
            return await _userService.UpdateUserProfile(userId, userProfile);
        }

        [HttpPut(nameof(UpdatePicture))]
        public async Task<UserProfile> UpdatePicture([FromBody] UserPicture userPicture)
        {
            var userId = _signInManager.UserManager.GetUserId(_signInManager.Context.User);
            return await _userService.UpdateUserPicture(userId, userPicture.Picture);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace AlgoMe.Controllers
{
    [Authorize]
    [ApiController]
    [Route("[controller]")]
    public class WeatherForecastController : ControllerBase
    {
        private static readonly string[] Summaries = new[]
        {
            "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering",
"Scorching"
        };

        private readonly ILogger<WeatherForecastController> _logger;

        public WeatherForecastController(ILogger<WeatherForecastController> logger)
        {
            _logger = logger;
        }

        [HttpGet]
        public IEnumerable<WeatherForecast> Get()
```

```
        {
            var rng = new Random();
            return Enumerable.Range(1, 5).Select(index => new WeatherForecast
            {
                Date = DateTime.Now.AddDays(index),
                TemperatureC = rng.Next(-20, 55),
                Summary = Summaries[rng.Next(Summaries.Length)]
            })
            .ToArray();
        }
    }
}
using AlgoMe.Models;
using IdentityServer4.EntityFramework.Options;
using Microsoft.AspNetCore.ApiAuthorization.IdentityServer;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AlgoMe.Data
{
    public class ApplicationDbContext : ApiAuthorizationDbContext<ApplicationUser>
    {
        public DbSet<Sector> Sector { get; set; }
        public DbSet<MarketCap> MarketCap { get; set; }
        public DbSet<InvestTransaction> InvestTransaction { get; set; }
        public DbSet<Portfolio> Portfolio { get; set; }
        public DbSet<PortfolioSector> PortfolioSector { get; set; }
        public DbSet<Risk> Risk { get; set; }
        public DbSet<Stock> Stock { get; set; }
        public DbSet<StockPrice> StockPrice { get; set; }
        public DbSet<PortfolioStock> PortfolioStocks { get; set; }
        public DbSet<CardAccount> CardAccount { get; set; }
        public ApplicationDbContext(
            DbContextOptions options,
            IOptions<OperationalStoreOptions> operationalStoreOptions) : base(options,
operationalStoreOptions)
        {
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
```

```
            base.OnModelCreating(builder);

            builder.Entity<PortfolioSector>()
                .HasKey(c => new {c.PortfolioId, c.SectorId});
        }
    }
}
using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace AlgoMe.Migrations
{
    public partial class InitialSchema : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(nullable: false),
                    Name = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(maxLength: 256, nullable: true),
                    ConcurrencyStamp = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table => new
                {
                    Id = table.Column<string>(nullable: false),
                    UserName = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedUserName = table.Column<string>(maxLength: 256, nullable: true),
                    Email = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedEmail = table.Column<string>(maxLength: 256, nullable: true),
                    EmailConfirmed = table.Column<bool>(nullable: false),
                    PasswordHash = table.Column<string>(nullable: true),
                    SecurityStamp = table.Column<string>(nullable: true),
                    ConcurrencyStamp = table.Column<string>(nullable: true),
                    PhoneNumber = table.Column<string>(nullable: true),
                    PhoneNumberConfirmed = table.Column<bool>(nullable: false),
```

```
            TwoFactorEnabled = table.Column<bool>(nullable: false),
            LockoutEnd = table.Column<DateTimeOffset>(nullable: true),
            LockoutEnabled = table.Column<bool>(nullable: false),
            AccessFailedCount = table.Column<int>(nullable: false),
            FirstName = table.Column<string>(nullable: true),
            LastName = table.Column<string>(nullable: true),
            Picture = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_AspNetUsers", x => x.Id);
        });

    migrationBuilder.CreateTable(
        name: "DeviceCodes",
        columns: table => new
        {
            UserCode = table.Column<string>(maxLength: 200, nullable: false),
            DeviceCode = table.Column<string>(maxLength: 200, nullable: false),
            SubjectId = table.Column<string>(maxLength: 200, nullable: true),
            ClientId = table.Column<string>(maxLength: 200, nullable: false),
            CreationTime = table.Column<DateTime>(nullable: false),
            Expiration = table.Column<DateTime>(nullable: false),
            Data = table.Column<string>(maxLength: 50000, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_DeviceCodes", x => x.UserCode);
        });

    migrationBuilder.CreateTable(
        name: "PersistedGrants",
        columns: table => new
        {
            Key = table.Column<string>(maxLength: 200, nullable: false),
            Type = table.Column<string>(maxLength: 50, nullable: false),
            SubjectId = table.Column<string>(maxLength: 200, nullable: true),
            ClientId = table.Column<string>(maxLength: 200, nullable: false),
            CreationTime = table.Column<DateTime>(nullable: false),
            Expiration = table.Column<DateTime>(nullable: true),
            Data = table.Column<string>(maxLength: 50000, nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_PersistedGrants", x => x.Key);
```

```
        });

    migrationBuilder.CreateTable(
        name: "Risk",
        columns: table => new
        {
            Id = table.Column<string>(nullable: false),
            Name = table.Column<string>(nullable: true),
            Description = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Risk", x => x.Id);
        });

    migrationBuilder.CreateTable(
        name: "Sector",
        columns: table => new
        {
            Id = table.Column<string>(nullable: false),
            Name = table.Column<string>(nullable: true),
            Description = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Sector", x => x.Id);
        });

    migrationBuilder.CreateTable(
        name: "Stock",
        columns: table => new
        {
            Id = table.Column<string>(nullable: false),
            Name = table.Column<string>(nullable: true),
            Description = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Stock", x => x.Id);
        });

    migrationBuilder.CreateTable(
        name: "AspNetRoleClaims",
        columns: table => new
        {
```

```
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            RoleId = table.Column<string>(nullable: false),
            ClaimType = table.Column<string>(nullable: true),
            ClaimValue = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_AspNetRoleClaims", x => x.Id);
            table.ForeignKey(
                name: "FK_AspNetRoleClaims_AspNetRoles_RoleId",
                column: x => x.RoleId,
                principalTable: "AspNetRoles",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "AspNetUserClaims",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            UserId = table.Column<string>(nullable: false),
            ClaimType = table.Column<string>(nullable: true),
            ClaimValue = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_AspNetUserClaims", x => x.Id);
            table.ForeignKey(
                name: "FK_AspNetUserClaims_AspNetUsers_UserId",
                column: x => x.UserId,
                principalTable: "AspNetUsers",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "AspNetUserLogins",
        columns: table => new
        {
            LoginProvider = table.Column<string>(nullable: false),
            ProviderKey = table.Column<string>(nullable: false),
            ProviderDisplayName = table.Column<string>(nullable: true),
```

```
            UserId = table.Column<string>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_AspNetUserLogins", x => new { x.LoginProvider, x.ProviderKey });
            table.ForeignKey(
                name: "FK_AspNetUserLogins_AspNetUsers_UserId",
                column: x => x.UserId,
                principalTable: "AspNetUsers",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "AspNetUserRoles",
        columns: table => new
        {
            UserId = table.Column<string>(nullable: false),
            RoleId = table.Column<string>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_AspNetUserRoles", x => new { x.UserId, x.RoleId });
            table.ForeignKey(
                name: "FK_AspNetUserRoles_AspNetRoles_RoleId",
                column: x => x.RoleId,
                principalTable: "AspNetRoles",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
            table.ForeignKey(
                name: "FK_AspNetUserRoles_AspNetUsers_UserId",
                column: x => x.UserId,
                principalTable: "AspNetUsers",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "AspNetUserTokens",
        columns: table => new
        {
            UserId = table.Column<string>(nullable: false),
            LoginProvider = table.Column<string>(nullable: false),
            Name = table.Column<string>(nullable: false),
            Value = table.Column<string>(nullable: true)
```

```
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetUserTokens", x => new { x.UserId, x.LoginProvider, x.Name
});
                table.ForeignKey(
                    name: "FK_AspNetUserTokens_AspNetUsers_UserId",
                    column: x => x.UserId,
                    principalTable: "AspNetUsers",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Cascade);
            });

        migrationBuilder.CreateTable(
            name: "Portfolio",
            columns: table => new
            {
                Id = table.Column<string>(nullable: false),
                UserId = table.Column<string>(nullable: true),
                InvestPeriod = table.Column<TimeSpan>(nullable: false),
                StartedTime = table.Column<DateTime>(nullable: false),
                RiskId = table.Column<string>(nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Portfolio", x => x.Id);
                table.ForeignKey(
                    name: "FK_Portfolio_Risk_RiskId",
                    column: x => x.RiskId,
                    principalTable: "Risk",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Restrict);
                table.ForeignKey(
                    name: "FK_Portfolio_AspNetUsers_UserId",
                    column: x => x.UserId,
                    principalTable: "AspNetUsers",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Restrict);
            });

        migrationBuilder.CreateTable(
            name: "StockPrice",
            columns: table => new
            {
                Id = table.Column<string>(nullable: false),
```

```
            Price = table.Column<decimal>(nullable: false),
            Time = table.Column<DateTime>(nullable: false),
            StockId = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_StockPrice", x => x.Id);
            table.ForeignKey(
                name: "FK_StockPrice_Stock_StockId",
                column: x => x.StockId,
                principalTable: "Stock",
                principalColumn: "Id",
                onDelete: ReferentialAction.Restrict);
        });

    migrationBuilder.CreateTable(
        name: "StockSector",
        columns: table => new
        {
            StockId = table.Column<string>(nullable: false),
            SectorId = table.Column<string>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_StockSector", x => new { x.SectorId, x.StockId });
            table.ForeignKey(
                name: "FK_StockSector_Sector_SectorId",
                column: x => x.SectorId,
                principalTable: "Sector",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
            table.ForeignKey(
                name: "FK_StockSector_Stock_StockId",
                column: x => x.StockId,
                principalTable: "Stock",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "InvestTransaction",
        columns: table => new
        {
            PortfolioId = table.Column<string>(nullable: false),
            StockId = table.Column<string>(nullable: false),
```

```
            Amount = table.Column<decimal>(nullable: false),
            Time = table.Column<DateTime>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_InvestTransaction", x => new { x.PortfolioId, x.StockId });
            table.ForeignKey(
                name: "FK_InvestTransaction_Portfolio_PortfolioId",
                column: x => x.PortfolioId,
                principalTable: "Portfolio",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
            table.ForeignKey(
                name: "FK_InvestTransaction_Stock_StockId",
                column: x => x.StockId,
                principalTable: "Stock",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateTable(
        name: "PortfolioSector",
        columns: table => new
        {
            PortfolioId = table.Column<string>(nullable: false),
            SectorId = table.Column<string>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_PortfolioSector", x => new { x.PortfolioId, x.SectorId });
            table.ForeignKey(
                name: "FK_PortfolioSector_Portfolio_PortfolioId",
                column: x => x.PortfolioId,
                principalTable: "Portfolio",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
            table.ForeignKey(
                name: "FK_PortfolioSector_Sector_SectorId",
                column: x => x.SectorId,
                principalTable: "Sector",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateIndex(
```

```
        name: "IX_AspNetRoleClaims_RoleId",
        table: "AspNetRoleClaims",
        column: "RoleId");

    migrationBuilder.CreateIndex(
        name: "RoleNameIndex",
        table: "AspNetRoles",
        column: "NormalizedName",
        unique: true,
        filter: "[NormalizedName] IS NOT NULL");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserClaims_UserId",
        table: "AspNetUserClaims",
        column: "UserId");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserLogins_UserId",
        table: "AspNetUserLogins",
        column: "UserId");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserRoles_RoleId",
        table: "AspNetUserRoles",
        column: "RoleId");

    migrationBuilder.CreateIndex(
        name: "EmailIndex",
        table: "AspNetUsers",
        column: "NormalizedEmail");

    migrationBuilder.CreateIndex(
        name: "UserNameIndex",
        table: "AspNetUsers",
        column: "NormalizedUserName",
        unique: true,
        filter: "[NormalizedUserName] IS NOT NULL");

    migrationBuilder.CreateIndex(
        name: "IX_DeviceCodes_DeviceCode",
        table: "DeviceCodes",
        column: "DeviceCode",
        unique: true);

    migrationBuilder.CreateIndex(
```

```
            name: "IX_DeviceCodes_Expiration",
            table: "DeviceCodes",
            column: "Expiration");

        migrationBuilder.CreateIndex(
            name: "IX_InvestTransaction_StockId",
            table: "InvestTransaction",
            column: "StockId");

        migrationBuilder.CreateIndex(
            name: "IX_PersistedGrants_Expiration",
            table: "PersistedGrants",
            column: "Expiration");

        migrationBuilder.CreateIndex(
            name: "IX_PersistedGrants_SubjectId_ClientId_Type",
            table: "PersistedGrants",
            columns: new[] { "SubjectId", "ClientId", "Type" });

        migrationBuilder.CreateIndex(
            name: "IX_Portfolio_RiskId",
            table: "Portfolio",
            column: "RiskId");

        migrationBuilder.CreateIndex(
            name: "IX_Portfolio_UserId",
            table: "Portfolio",
            column: "UserId");

        migrationBuilder.CreateIndex(
            name: "IX_PortfolioSector_SectorId",
            table: "PortfolioSector",
            column: "SectorId");

        migrationBuilder.CreateIndex(
            name: "IX_StockPrice_StockId",
            table: "StockPrice",
            column: "StockId");

        migrationBuilder.CreateIndex(
            name: "IX_StockSector_StockId",
            table: "StockSector",
            column: "StockId");
    }
```

```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "AspNetRoleClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserLogins");

    migrationBuilder.DropTable(
        name: "AspNetUserRoles");

    migrationBuilder.DropTable(
        name: "AspNetUserTokens");

    migrationBuilder.DropTable(
        name: "DeviceCodes");

    migrationBuilder.DropTable(
        name: "InvestTransaction");

    migrationBuilder.DropTable(
        name: "PersistedGrants");

    migrationBuilder.DropTable(
        name: "PortfolioSector");

    migrationBuilder.DropTable(
        name: "StockPrice");

    migrationBuilder.DropTable(
        name: "StockSector");

    migrationBuilder.DropTable(
        name: "AspNetRoles");

    migrationBuilder.DropTable(
        name: "Portfolio");

    migrationBuilder.DropTable(
        name: "Sector");

    migrationBuilder.DropTable(
```

```
            name: "Stock");

        migrationBuilder.DropTable(
            name: "Risk");

        migrationBuilder.DropTable(
            name: "AspNetUsers");
    }
  }
}
// <auto-generated />
using System;
using AlgoMe.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

namespace AlgoMe.Migrations
{
  [DbContext(typeof(ApplicationDbContext))]
  [Migration("20200523140613_InitialSchema")]
  partial class InitialSchema
  {
    protected override void BuildTargetModel(ModelBuilder modelBuilder)
    {
#pragma warning disable 612, 618
        modelBuilder
          .HasAnnotation("ProductVersion", "3.1.4")
          .HasAnnotation("Relational:MaxIdentifierLength", 128)
          .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

        modelBuilder.Entity("AlgoMe.Models.ApplicationUser", b =>
          {
            b.Property<string>("Id")
              .HasColumnType("nvarchar(450)");

            b.Property<int>("AccessFailedCount")
              .HasColumnType("int");

            b.Property<string>("ConcurrencyStamp")
              .IsConcurrencyToken()
              .HasColumnType("nvarchar(max)");
```

```
b.Property<string>("Email")
   .HasColumnType("nvarchar(256)")
   .HasMaxLength(256);

b.Property<bool>("EmailConfirmed")
   .HasColumnType("bit");

b.Property<string>("FirstName")
   .HasColumnType("nvarchar(max)");

b.Property<string>("LastName")
   .HasColumnType("nvarchar(max)");

b.Property<bool>("LockoutEnabled")
   .HasColumnType("bit");

b.Property<DateTimeOffset?>("LockoutEnd")
   .HasColumnType("datetimeoffset");

b.Property<string>("NormalizedEmail")
   .HasColumnType("nvarchar(256)")
   .HasMaxLength(256);

b.Property<string>("NormalizedUserName")
   .HasColumnType("nvarchar(256)")
   .HasMaxLength(256);

b.Property<string>("PasswordHash")
   .HasColumnType("nvarchar(max)");

b.Property<string>("PhoneNumber")
   .HasColumnType("nvarchar(max)");

b.Property<bool>("PhoneNumberConfirmed")
   .HasColumnType("bit");

b.Property<string>("Picture")
   .HasColumnType("nvarchar(max)");

b.Property<string>("SecurityStamp")
   .HasColumnType("nvarchar(max)");

b.Property<bool>("TwoFactorEnabled")
   .HasColumnType("bit");
```

```
            b.Property<string>("UserName")
                .HasColumnType("nvarchar(256)")
                .HasMaxLength(256);

            b.HasKey("Id");

            b.HasIndex("NormalizedEmail")
                .HasName("EmailIndex");

            b.HasIndex("NormalizedUserName")
                .IsUnique()
                .HasName("UserNameIndex")
                .HasFilter("[NormalizedUserName] IS NOT NULL");

            b.ToTable("AspNetUsers");
        });

    modelBuilder.Entity("AlgoMe.Models.InvestTransaction", b =>
        {
            b.Property<string>("PortfolioId")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("StockId")
                .HasColumnType("nvarchar(450)");

            b.Property<decimal>("Amount")
                .HasColumnType("decimal(18,2)");

            b.Property<DateTime>("Time")
                .HasColumnType("datetime2");

            b.HasKey("PortfolioId", "StockId");

            b.HasIndex("StockId");

            b.ToTable("InvestTransaction");
        });

    modelBuilder.Entity("AlgoMe.Models.Portfolio", b =>
        {
            b.Property<string>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("nvarchar(450)");
```

```
            b.Property<TimeSpan>("InvestPeriod")
                .HasColumnType("time");

            b.Property<string>("RiskId")
                .HasColumnType("nvarchar(450)");

            b.Property<DateTime>("StartedTime")
                .HasColumnType("datetime2");

            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");

            b.HasKey("Id");

            b.HasIndex("RiskId");

            b.HasIndex("UserId");

            b.ToTable("Portfolio");
        });

    modelBuilder.Entity("AlgoMe.Models.PortfolioSector", b =>
        {
            b.Property<string>("PortfolioId")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("SectorId")
                .HasColumnType("nvarchar(450)");

            b.HasKey("PortfolioId", "SectorId");

            b.HasIndex("SectorId");

            b.ToTable("PortfolioSector");
        });

    modelBuilder.Entity("AlgoMe.Models.Risk", b =>
        {
            b.Property<string>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Description")
                .HasColumnType("nvarchar(max)");
```

```
            b.Property<string>("Name")
                .HasColumnType("nvarchar(max)");

            b.HasKey("Id");

            b.ToTable("Risk");
        });

    modelBuilder.Entity("AlgoMe.Models.Sector", b =>
        {
            b.Property<string>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Description")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("Name")
                .HasColumnType("nvarchar(max)");

            b.HasKey("Id");

            b.ToTable("Sector");
        });

    modelBuilder.Entity("AlgoMe.Models.Stock", b =>
        {
            b.Property<string>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Description")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("Name")
                .HasColumnType("nvarchar(max)");

            b.HasKey("Id");

            b.ToTable("Stock");
        });

    modelBuilder.Entity("AlgoMe.Models.StockPrice", b =>
        {
            b.Property<string>("Id")
```

```
            .ValueGeneratedOnAdd()
            .HasColumnType("nvarchar(450)");

        b.Property<decimal>("Price")
            .HasColumnType("decimal(18,2)");

        b.Property<string>("StockId")
            .HasColumnType("nvarchar(450)");

        b.Property<DateTime>("Time")
            .HasColumnType("datetime2");

        b.HasKey("Id");

        b.HasIndex("StockId");

        b.ToTable("StockPrice");
    });

modelBuilder.Entity("AlgoMe.Models.StockSector", b =>
    {
        b.Property<string>("SectorId")
            .HasColumnType("nvarchar(450)");

        b.Property<string>("StockId")
            .HasColumnType("nvarchar(450)");

        b.HasKey("SectorId", "StockId");

        b.HasIndex("StockId");

        b.ToTable("StockSector");
    });

modelBuilder.Entity("IdentityServer4.EntityFramework.Entities.DeviceFlowCodes", b =>
    {
        b.Property<string>("UserCode")
            .HasColumnType("nvarchar(200)")
            .HasMaxLength(200);

        b.Property<string>("ClientId")
            .IsRequired()
            .HasColumnType("nvarchar(200)")
            .HasMaxLength(200);
```

```
            b.Property<DateTime>("CreationTime")
                .HasColumnType("datetime2");

            b.Property<string>("Data")
                .IsRequired()
                .HasColumnType("nvarchar(max)")
                .HasMaxLength(50000);

            b.Property<string>("DeviceCode")
                .IsRequired()
                .HasColumnType("nvarchar(200)")
                .HasMaxLength(200);

            b.Property<DateTime?>("Expiration")
                .IsRequired()
                .HasColumnType("datetime2");

            b.Property<string>("SubjectId")
                .HasColumnType("nvarchar(200)")
                .HasMaxLength(200);

            b.HasKey("UserCode");

            b.HasIndex("DeviceCode")
                .IsUnique();

            b.HasIndex("Expiration");

            b.ToTable("DeviceCodes");
        });

    modelBuilder.Entity("IdentityServer4.EntityFramework.Entities.PersistedGrant", b =>
        {
            b.Property<string>("Key")
                .HasColumnType("nvarchar(200)")
                .HasMaxLength(200);

            b.Property<string>("ClientId")
                .IsRequired()
                .HasColumnType("nvarchar(200)")
                .HasMaxLength(200);

            b.Property<DateTime>("CreationTime")
                .HasColumnType("datetime2");
```

```
            b.Property<string>("Data")
                .IsRequired()
                .HasColumnType("nvarchar(max)")
                .HasMaxLength(50000);

            b.Property<DateTime?>("Expiration")
                .HasColumnType("datetime2");

            b.Property<string>("SubjectId")
                .HasColumnType("nvarchar(200)")
                .HasMaxLength(200);

            b.Property<string>("Type")
                .IsRequired()
                .HasColumnType("nvarchar(50)")
                .HasMaxLength(50);

            b.HasKey("Key");

            b.HasIndex("Expiration");

            b.HasIndex("SubjectId", "ClientId", "Type");

            b.ToTable("PersistedGrants");
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("ConcurrencyStamp")
                .IsConcurrencyToken()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("Name")
                .HasColumnType("nvarchar(256)")
                .HasMaxLength(256);

            b.Property<string>("NormalizedName")
                .HasColumnType("nvarchar(256)")
                .HasMaxLength(256);

            b.HasKey("Id");
```

```
            b.HasIndex("NormalizedName")
                .IsUnique()
                .HasName("RoleNameIndex")
                .HasFilter("[NormalizedName] IS NOT NULL");

            b.ToTable("AspNetRoles");
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b =>
        {
            b.Property<int>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("int")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            b.Property<string>("ClaimType")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("ClaimValue")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("RoleId")
                .IsRequired()
                .HasColumnType("nvarchar(450)");

            b.HasKey("Id");

            b.HasIndex("RoleId");

            b.ToTable("AspNetRoleClaims");
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b =>
        {
            b.Property<int>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("int")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            b.Property<string>("ClaimType")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("ClaimValue")
```

```
            .HasColumnType("nvarchar(max)");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("nvarchar(450)");

        b.HasKey("Id");

        b.HasIndex("UserId");

        b.ToTable("AspNetUserClaims");
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>", b =>
    {
        b.Property<string>("LoginProvider")
            .HasColumnType("nvarchar(450)");

        b.Property<string>("ProviderKey")
            .HasColumnType("nvarchar(450)");

        b.Property<string>("ProviderDisplayName")
            .HasColumnType("nvarchar(max)");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("nvarchar(450)");

        b.HasKey("LoginProvider", "ProviderKey");

        b.HasIndex("UserId");

        b.ToTable("AspNetUserLogins");
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>", b =>
    {
        b.Property<string>("UserId")
            .HasColumnType("nvarchar(450)");

        b.Property<string>("RoleId")
            .HasColumnType("nvarchar(450)");

        b.HasKey("UserId", "RoleId");
```

```
            b.HasIndex("RoleId");

            b.ToTable("AspNetUserRoles");
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>", b =>
        {
            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("LoginProvider")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Name")
                .HasColumnType("nvarchar(450)");

            b.Property<string>("Value")
                .HasColumnType("nvarchar(max)");

            b.HasKey("UserId", "LoginProvider", "Name");

            b.ToTable("AspNetUserTokens");
        });

    modelBuilder.Entity("AlgoMe.Models.InvestTransaction", b =>
        {
            b.HasOne("AlgoMe.Models.Portfolio", "Portfolio")
                .WithMany()
                .HasForeignKey("PortfolioId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("AlgoMe.Models.Stock", "Stock")
                .WithMany()
                .HasForeignKey("StockId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("AlgoMe.Models.Portfolio", b =>
        {
            b.HasOne("AlgoMe.Models.Risk", "Risk")
                .WithMany()
                .HasForeignKey("RiskId");
```

```
            b.HasOne("AlgoMe.Models.ApplicationUser", "User")
                .WithMany()
                .HasForeignKey("UserId");
        });

    modelBuilder.Entity("AlgoMe.Models.PortfolioSector", b =>
        {
            b.HasOne("AlgoMe.Models.Portfolio", "Portfolio")
                .WithMany()
                .HasForeignKey("PortfolioId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("AlgoMe.Models.Sector", "Sector")
                .WithMany()
                .HasForeignKey("SectorId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("AlgoMe.Models.StockPrice", b =>
        {
            b.HasOne("AlgoMe.Models.Stock", "Stock")
                .WithMany()
                .HasForeignKey("StockId");
        });

    modelBuilder.Entity("AlgoMe.Models.StockSector", b =>
        {
            b.HasOne("AlgoMe.Models.Sector", "Sector")
                .WithMany()
                .HasForeignKey("SectorId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("AlgoMe.Models.Stock", "Stock")
                .WithMany()
                .HasForeignKey("StockId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b =>
        {
            b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
```

```
            .WithMany()
            .HasForeignKey("RoleId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b =>
        {
            b.HasOne("AlgoMe.Models.ApplicationUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>", b =>
        {
            b.HasOne("AlgoMe.Models.ApplicationUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>", b =>
        {
            b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
                .WithMany()
                .HasForeignKey("RoleId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("AlgoMe.Models.ApplicationUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });

    modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>", b =>
        {
            b.HasOne("AlgoMe.Models.ApplicationUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
```

```
                .IsRequired();
            });
#pragma warning restore 612, 618
        }
    }
}
using Microsoft.EntityFrameworkCore.Migrations;

namespace AlgoMe.Migrations
{
    public partial class AddedMarketCap : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<int>(
                name: "MarketCap",
                table: "Stock",
                nullable: false,
                defaultValue: 0);
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "MarketCap",
                table: "Stock");
        }
    }
}


using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace AlgoMe.Pages
{
    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public class ErrorModel : PageModel
    {
```

```
        private readonly ILogger<ErrorModel> _logger;

        public ErrorModel(ILogger<ErrorModel> logger)
        {
            _logger = logger;
        }

        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

        public void OnGet()
        {
            RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;
        }
    }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
    public class GlobalRepository
    {
        public Repository<MarketCap> MarketCapRepository { get; }
        public Repository<Stock> StockRepository { get; }
        public Repository<Sector> SectorRepository { get; }
        public Repository<Portfolio> PortfolioRepository { get; }
        public Repository<ApplicationUser> UserRepository { get; }
        public Repository<PortfolioStock> PortfolioStockRepository { get; }
        public Repository<CardAccount> CardAccountRepository { get; }
        public Repository<InvestTransaction> InvestTransactionRepository { get; }

        public GlobalRepository(ApplicationDbContext context)
        {
            MarketCapRepository = new Repository<MarketCap>(context);
            StockRepository = new Repository<Stock>(context);
            SectorRepository = new Repository<Sector>(context);
            PortfolioRepository = new Repository<Portfolio>(context);
            PortfolioStockRepository = new Repository<PortfolioStock>(context);
            UserRepository = new Repository<ApplicationUser>(context);
            CardAccountRepository = new Repository<CardAccount>(context);
            InvestTransactionRepository = new Repository<InvestTransaction>(context);
        }
    }
```

```
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore.ChangeTracking;

namespace AlgoMe.Repositories
{
   public interface IRepository<T> where T : class
   {
      IQueryable<T> GetAll();
      Task<T> GetById(string id);
      Task<EntityEntry<T>> Create(T obj);
      Task SaveChanges();
      EntityEntry<T> Remove(T entity);
   }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
   public class MarketCapacitiesRepository : Repository<MarketCap>
   {
      protected MarketCapacitiesRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
      {
      }
   }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
   public class PortfolioRepository : Repository<Portfolio>
   {
      public PortfolioRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
      {
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
using AlgoMe.Data;
```

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.ChangeTracking;

namespace AlgoMe.Repositories
{
   public class Repository<T> : IRepository<T> where T : class
   {
      private readonly ApplicationDbContext _applicationDbContext;

      public Repository(ApplicationDbContext applicationDbContext)
      {
         _applicationDbContext = applicationDbContext;
      }

      public IQueryable<T> GetAll()
      {
         return _applicationDbContext.Set<T>().AsQueryable().AsNoTracking();
      }

      public Task<T> GetById(string id)
      {
         return _applicationDbContext.FindAsync<T>(id).AsTask();
      }

      public Task<EntityEntry<T>> Create(T obj)
      {
         return _applicationDbContext.AddAsync(obj).AsTask();
      }

      public Task SaveChanges()
      {
         return _applicationDbContext.SaveChangesAsync();
      }

      public EntityEntry<T> Remove(T entity)
      {
         return _applicationDbContext.Remove(entity);
      }
   }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
```

```
    public class SectorRepository : Repository<Sector>
    {
        protected SectorRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
        }
    }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
    public class StockRepository : Repository<Stock>
    {
        protected StockRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
        }
    }
}
using AlgoMe.Data;
using AlgoMe.Models;

namespace AlgoMe.Repositories
{
    public class UserRepository : Repository<ApplicationUser>
    {
        public UserRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
        }
    }
}
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.ViewModel;

namespace AlgoMe.Services.Interfaces
{
    public interface ICardService
    {
        Task<CardAccount[]> GetCardAccounts(string userId);
        Task<CardAccount> CreateCardAccount(string userId, CardViewModel cardViewModel);
        Task RemoveCardAccount(string userId, string cardAccountId);
```

```
    }
}
using System.Collections.Generic;
using System.Threading.Tasks;
using AlgoMe.Models;

namespace AlgoMe.Services.Interfaces
{
    public interface ICompaniesService
    {
        Task<List<Stock>> GetStocks(List<string> sectors, List<string> marketCapacities, Portfolio
portfolio);
        Task<List<Sector>> GetSectors();
        Task<List<MarketCap>> GetMarketCapacities();
    }
}
using System.Threading.Tasks;
using AlgoMe.Controllers;
using AlgoMe.Models;

namespace AlgoMe.Services.Interfaces
{
    public interface IPortfolioService
    {
        Task<Portfolio> GetByUserId(string userId);
        Task<PortfolioStock> AddCompany(string userId, string stockId);
        Task DeleteCompany(string userId, string stockId);
        Task<InvestTransaction> MakeOperation(OperationType operationType, OperationViewModel
operationViewModel,
            string userId);

        Task<InvestTransaction[]> GetTransactions(string userId);
        Task<decimal> GetEarnings(string userId);
    }
}
using System.Threading.Tasks;
using AlgoMe.Models;

namespace AlgoMe.Services.Interfaces
{
    public interface IPredictionService
    {
        public Task<PredictionModel> GetPrediction(string stockId, string userId);
    }
}
```

```
using System.Threading.Tasks;
using AlphaVantage.Net.Stocks.TimeSeries;
using Microsoft.EntityFrameworkCore;

namespace AlgoMe.Services.Interfaces
{
   public interface IStockService
   {
      Task<StockTimeSeries> GetTimeSeries(string stockId);
      Task<decimal> GetCurrentPrice(string stockId);
      Task<int> GetAmount(string stockId, string userId);
   }
}
using System.Threading.Tasks;
using AlgoMe.ViewModel;

namespace AlgoMe.Services.Interfaces
{
   public interface IUserProfileService
   {
      Task<UserProfile> GetUserProfileById(string id);
      Task<UserProfile> UpdateUserProfile(string userId, UserProfile userProfile);
      Task<UserProfile> UpdateUserPicture(string userId, string picture);
   }
}
using System.Threading.Tasks;
using AlgoMe.ViewModel;

namespace AlgoMe.Services.Interfaces
{
   public interface IUserService
   {
      Task<TokenViewModel> Register(RegisterViewModel registerViewModel);
      Task<TokenViewModel> Login(LoginViewModel loginViewModel);
      Task Reset(ResetViewModel resetViewModel);
      Task Logout();

   }
}
using System;
using System.Linq;
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Repositories;
using AlgoMe.Services.Interfaces;
```

```
using AlgoMe.ViewModel;
using Microsoft.EntityFrameworkCore;

namespace AlgoMe.Services
{
   public class CardService : ICardService
   {
      private readonly GlobalRepository _globalRepository;

      public CardService(GlobalRepository globalRepository)
      {
         _globalRepository = globalRepository;
      }

      public async Task<CardAccount[]> GetCardAccounts(string userId)
      {
         var portfolio = await _globalRepository.PortfolioRepository.GetAll()
            .FirstOrDefaultAsync(item => item.UserId == userId);
         var portfolioId = portfolio.Id;
         return await _globalRepository.CardAccountRepository.GetAll()
            .Where(item => !item.IsDisabled && item.PortfolioId == portfolioId)
            .ToArrayAsync();
      }

      public async Task<CardAccount> CreateCardAccount(string userId, CardViewModel
cardViewModel)
      {
         var portfolio = await _globalRepository.PortfolioRepository.GetAll()
            .FirstOrDefaultAsync(item => item.UserId == userId);

         var cardAccount = new CardAccount
         {
            Name = cardViewModel.Name,
            CardNumber = cardViewModel.CardNumber,
            AddedDate = DateTime.UtcNow,
            PortfolioId = portfolio.Id
         };

         var entityCard = await _globalRepository.CardAccountRepository.Create(cardAccount);
         await _globalRepository.CardAccountRepository.SaveChanges();
         return entityCard.Entity;
      }

      public async Task RemoveCardAccount(string userId, string cardAccountId)
      {
```

```
            var portfolio = await _globalRepository.PortfolioRepository.GetAll()
                .FirstOrDefaultAsync(item => item.UserId == userId);
            var portfolioId = portfolio.Id;
            var cardAccount = await _globalRepository.CardAccountRepository.GetById(cardAccountId);
            if (cardAccount.PortfolioId == portfolioId)
            {
                cardAccount.IsDisabled = true;
                await _globalRepository.CardAccountRepository.SaveChanges();
            }
        }
    }
}
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AlgoMe.Models;
using AlgoMe.Repositories;
using AlgoMe.Services.Interfaces;
using IdentityServer4.Extensions;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Internal;

namespace AlgoMe.Services
{
    public class CompaniesService : ICompaniesService
    {
        private readonly Repository<MarketCap> _marketCapacitiesRepository;
        private readonly Repository<Stock> _stockRepository;
        private readonly Repository<Sector> _sectorRepository;

        public CompaniesService(GlobalRepository globalRepository)
        {
            _marketCapacitiesRepository = globalRepository.MarketCapRepository;
            _stockRepository = globalRepository.StockRepository;
            _sectorRepository = globalRepository.SectorRepository;
        }

        public async Task<List<Stock>> GetStocks(List<string> sectors, List<string> marketCapacities,
            Portfolio portfolio)
        {//TODO: fix all bad things
            var stocks = _stockRepository.GetAll();
            if (sectors != null && sectors.Any())
                stocks = stocks.Where(stock => sectors.Contains(stock.SectorId));

            if (marketCapacities != null && marketCapacities.Any())
```

```
        {
            var marketCaps = await _marketCapacitiesRepository.GetAll()
                .Where(mc => marketCapacities.Contains(mc.Id))
                .ToListAsync();

            stocks = stocks.AsEnumerable()
                .Where(stock =>
                    marketCaps.Any(mc => mc.Min < stock.MarketCap && stock.MarketCap <= mc.Max))
                .AsQueryable();
        }

        if (portfolio != null)
        {
            stocks = stocks.AsEnumerable()
                .Where(stock =>
                    !portfolio.PortfolioStocks.Any() ||
                    portfolio.PortfolioStocks.All(ps => ps.StockId != stock.Id))
                .AsQueryable();
        }

        return stocks.ToList();
    }

    public Task<List<Sector>> GetSectors()
    {
        return _sectorRepository.GetAll().ToListAsync();
    }

    public Task<List<MarketCap>> GetMarketCapacities()
    {
        return _marketCapacitiesRepository.GetAll().ToListAsync();
    }
  }
}
using System;
using System.Linq;
using System.Threading.Tasks;
using AlgoMe.Controllers;
using AlgoMe.Models;
using AlgoMe.Repositories;
using AlgoMe.Services.Interfaces;
using Microsoft.EntityFrameworkCore;

namespace AlgoMe.Services
{
```

```
public class PortfolioService : IPortfolioService
{
    private readonly GlobalRepository _globalRepository;
    private readonly IStockService _stockService;

    public PortfolioService(GlobalRepository globalRepository,
        IStockService stockService)
    {
        _globalRepository = globalRepository;
        _stockService = stockService;
    }

    public async Task<Portfolio> GetByUserId(string userId)
    {
        var portfolio = await _globalRepository.PortfolioRepository.GetAll()
            .Include(p => p.PortfolioStocks)
            .ThenInclude(p => p.Stock)
            .FirstOrDefaultAsync(item => item.UserId == userId);

        if (portfolio != null) return portfolio;

        portfolio = new Portfolio
        {
            UserId = userId, InvestPeriod = DateTime.UtcNow + TimeSpan.FromDays(10), StartedTime
= DateTime.UtcNow
        };
        var entity = await _globalRepository.PortfolioRepository.Create(portfolio);
        await _globalRepository.PortfolioRepository.SaveChanges();
        portfolio = entity.Entity;

        return portfolio;
    }

    public async Task<PortfolioStock> AddCompany(string userId, string stockId)
    {
        var portfolio = await _globalRepository.PortfolioRepository.GetAll()
            .FirstOrDefaultAsync(p => p.UserId == userId);

        var portfolioId = portfolio.Id;

        var ps = await _globalRepository.PortfolioStockRepository.GetAll()
            .FirstOrDefaultAsync(pp => pp.PortfolioId == portfolioId && pp.StockId == stockId);

        if (ps != null) return ps;
```

```
            ps = new PortfolioStock {PortfolioId = portfolioId, StockId = stockId};

            var psEntity = await _globalRepository.PortfolioStockRepository.Create(ps);
            await _globalRepository.PortfolioRepository.SaveChanges();
            return psEntity.Entity;
        }

        public async Task DeleteCompany(string userId, string stockId)
        {
            var portfolio = await _globalRepository.PortfolioRepository.GetAll()
                .FirstOrDefaultAsync(p => p.UserId == userId);

            var portfolioId = portfolio.Id;

            var ps = await _globalRepository.PortfolioStockRepository.GetAll()
                .FirstOrDefaultAsync(pp => pp.PortfolioId == portfolioId && pp.StockId == stockId);

            if (ps != null)
            {
                _globalRepository.PortfolioStockRepository.Remove(ps);
                await _globalRepository.PortfolioRepository.SaveChanges();
            }
        }

        public async Task<InvestTransaction> MakeOperation(OperationType operationType,
            OperationViewModel operationViewModel,
            string userId)
        {
            var portfolio = await _globalRepository.PortfolioRepository.GetAll()
                .Include(item => item.PortfolioStocks)
                .FirstOrDefaultAsync(p => p.UserId == userId);
            var cardAccount = await
    _globalRepository.CardAccountRepository.GetById(operationViewModel.AccountId);
            if (cardAccount.PortfolioId != portfolio.Id)
                throw new Exception("Card account is wrong");

            if (portfolio.PortfolioStocks.All(item => item.StockId != operationViewModel.StockId))
                throw new Exception("StockId is invalid");

            if(operationViewModel.Amount <= 0)
                throw new Exception("Amount should be greater then 0");

            var currentAmount = await _stockService.GetAmount(operationViewModel.StockId, userId);
            if(operationType == OperationType.Sell && currentAmount < operationViewModel.Amount)
                throw new Exception("Amount should be less or equals to your stock amount");
```

```
            var price = await _stockService.GetCurrentPrice(operationViewModel.StockId);

            var investTransaction = new InvestTransaction
            {
                Amount = operationViewModel.Amount,
                Price = price,
                StockId = operationViewModel.StockId,
                CardAccountId = operationViewModel.AccountId,
                Time = DateTime.UtcNow,
                OperationType = operationType
            };

            var entity = await _globalRepository.InvestTransactionRepository.Create(investTransaction);
            await _globalRepository.InvestTransactionRepository.SaveChanges();
            return entity.Entity;
        }

        public async Task<InvestTransaction[]> GetTransactions(string userId)
        {
            var portfolio = await _globalRepository.PortfolioRepository.GetAll()
                .FirstOrDefaultAsync(p => p.UserId == userId);

            var portfolioId = portfolio.Id;

            var transactions = await _globalRepository.InvestTransactionRepository.GetAll()
                .Include(item=>item.CardAccount)
                .Include(item=>item.Stock)
                .Where(item => item.CardAccount.PortfolioId == portfolioId)
                .OrderByDescending(item=>item.Time)
                .ToArrayAsync();

            return transactions;
        }

        public async Task<decimal> GetEarnings(string userId)
        {
            var transactions = await GetTransactions(userId);

            var sum = transactions
                .Sum(item => Math.Sign((int) item.OperationType) * item.Amount * item.Price);
            return sum;
        }
    }
}
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кафедра автоматизованих систем обробки інформації і управління

**УЗГОДЖЕНО**

**Керівник проєкту**

_____ *О.М. Клименко*

(підпис) (ініціали, прізвище)

“13” квітня 2020 р.

**ЗАТВЕРДЖУЮ**

**В.о. завідувача кафедри**

_____ *О.А. Павлов*

(підпис) (ініціали, прізвище)

“14” квітня 2020 р.

Інформаційна система моніторингу та аналізу біржових ринків

# ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6123.01.000 ТЗ*

на 9 сторінках

Київ – 2020 року

**ЗМІСТ**

# 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

## 1.1 Повне найменування системи та її умовне позначення

**Повна назва системи:** Web-застосування для моніторингу та аналізу біржових ринків.

## 1.2 Найменування організації-замовника та організацій-учасників робіт

Генеральним замовником проекту являється кафедра Автоматизованих систем обробки інформації та управління НТУУ "КПІ". Представником замовника є Клименко Олена Миколаївна.

Розробниками системи є студенти групи ІС-42 факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського» Проскурка Даниіл.

## 1.2 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

## 1.3 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи спілкування та автоматизованого пошуку місць зустрічі користувачів – 29 лютого 2020 рік.

Плановий термін по закінченню роботи над створенням системи спілкування та автоматизованого пошуку місць зустрічі користувачів – не пізніше 1 червня 2020 року.

# 2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

## 2.1 Призначення системи

Інформаційна система моніторингу та аналізу біржових ринків призначена для ефективного спостереження за станом ринку та допомоги у прийнятті рішення щодо інвестування.

## 2.2 Цілі створення системи

Метою даної інформаційної системи є підвищення прибуткованості від інвестування шляхом пошуку менш ризикованих шляхів та спостереженням зміни стану ринку, на основі яких створювати реомендації щодо продажу чи купівлі цінних паперів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

а) ведення аккаунту користувача

б) створення портфелю користувача

в) збір даних щодо стану ринку

г) обробка даних з ринку

д) створення моделі що відображає ринок

е) формування звітності

## 2 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Для користування сервісом обов’язковою умовою для користувача є наявність браузера, який підтримує JavaScript, адже даний сервіс є web-застосуванням.

Працювати з сервісом можна тільки авторизованим користувачам, які авторизуються через електронну адресу. Після авторизації користувач отримає доступ до настройки профілю та аналітики ринку.

Об’єктом автоматизації є прийнятя рішення щодо купівлі чи продажу акцій.

# 4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Вимоги до функціональних характеристик

Даний сервіс повинен задовольняти потреби користувачів, що цікавляться купівлею та продажем акцій. Сервіс має виконувати наступні функції:

- Створення користувачів – сервіс надає можливість створення аккаунту користувача та упраляти ним.
- Формування звіту – система формує звіт з аналізом ринку та пропозиціями щодо купівлі чи продажу акцій.

## 4.2 Вимоги до надійності

Програма повинна зберігати працездатність і забезпечувати відновлення своїх функцій при виникненні наступних позаштатних ситуацій:

– при помилках в роботі апаратних засобів (крім носіїв даних і програм). Відновлення функції програми покладається на хостинг;

– при помилках, пов’язаних з особливістю різноманітних браузерів, існуючих на сьогоднішній день. Реалізація кросбраузерності web-застосунку.

Програмний продукт повинен поєднувати надійність та функціональність. У разі виникнення аварійних ситуацій необхідно сповіщати користувача та надавати інструкцію для подальших дій. Будь-які аварійні ситуації мають бути задокументовані у звіті, який при необхідності надсилається розробнику для визначення причини збою в роботі та усуненні помилок, які могли привести до нестабільної роботи програмного продукту.

## 4.3 Умови експлуатації

Не передачає особливі умови єксплуатації.

## 4.4 Вимоги до складу і параметрів технічних засобів

Склад, структура і способи організації даних в системі повинні бути визначені на етапі технічного проектування.

Структура технічних засобів визначається виходячи із можливості їх забезпечити виконання встановлених операцій процесу технічного обслуговування.

Для правильної роботи розробленої системи до складу технічних засобів повинен входити комп’ютер, що має конфігурацію наведену нижче:

а) комп’ютер з такою конфігурацією:

1) процесор з тактовою частотою не нижче 1 ГГц;
2) об’єм оперативної пам’яті не менше 128 Мб;
3) інші складові можуть мати будь-які параметри, тому що вони не значним чином впливають на роботу застосування;
4) підключення до мережі інтернет;

б) додатково має бути встановлене таке програмне забезпечення:

1) процесор з тактовою частотою не нижче 1 ГГц;
2) об’єм оперативної пам’яті не менше 256 Мб;
3) інші складові можуть мати будь-які параметри, тому що вони не значним чином впливають на роботу програми;
4) наявність швидкісного інтернет-з’єднання;

в) комп’ютерна периферія, до складу якої входить:

1) монітор;
2) мишка;
3) клавіатура.

## 5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки системи ведення наукової роботи.

| № п/п | Назва етапу роботи | Термін виконання етапу | Результат виконання |
|---|---|---|---|
| 1. | Підготовка технічного завдання на розробку програмного продукту | 07.03.2020 | |
| 2. | Розробка сценарію роботи | 12.03.2020 | |
| 3. | Технічне проектування – функціональність, модулі, задачі, цілі тощо | 20.03.2020 | |
| 4. | Узгодження з керівником інтерфейсу користувача | 02.04.2020 | |
| 5. | Розробка інформаційного забезпечення | 17.04.2020 | |
| 6. | Розробка програмного забезпечення | 29.04.2020 | |
| 7. | Налагодження програми | 13.05.2020 | |
| 8. | Тестування програми | 27.05.2020 | |
| 9. | Здача готового програмного продукту замовнику | 01.06.2020 | |

# 6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

## 6.1 Види випробувань

Для контролю правильності роботи програмного забезпечення буде проведено функціональне тестування. В ході тестування буде проведено випробування основних функціональних характеристик системи та цілої системи загалом.

Тестування авторизації користувача полягає в представленні користувачу головної сторінки з відповідним повідомленням.

Тестування редагування профілю користувача полягає в представленні користувачу сторінки з відповідною формою та у використанні в подальшому нових даних, введених користувачем.

Тестування створення звіту полягає у представлення користувачу сторінки з аналітикою по запиту користувача.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1003979404

Дата перевірки:
12.06.2020 03:22:15 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
12.06.2020 13:57:17 EEST

ID користувача:
77149

Назва документу: Proskurka_is61_3

ID файлу: 1003994452 Кількість сторінок: 40 Кількість слів: 4522 Кількість символів: 34483 Розмір файлу: 1,022.64 KB

## 14% Схожість

Найбільша схожість: 2.85% з джерело бібліотеки. ID файлу: 1003665183

| | | |
|---|---|---|
| 4.11% Схожість з Інтернет джерелами | 45 | Page 42 |
| 13.8% Текстові збіги по Бібліотеці акаунту | 297 | Page 42 |

## 0.44% Цитат

| | | |
|---|---|---|
| Цитати | 1 | Page 43 |

Вилучення переліку посилань вимкнено

## 0% Вилучень

Вилучений текст відсутній

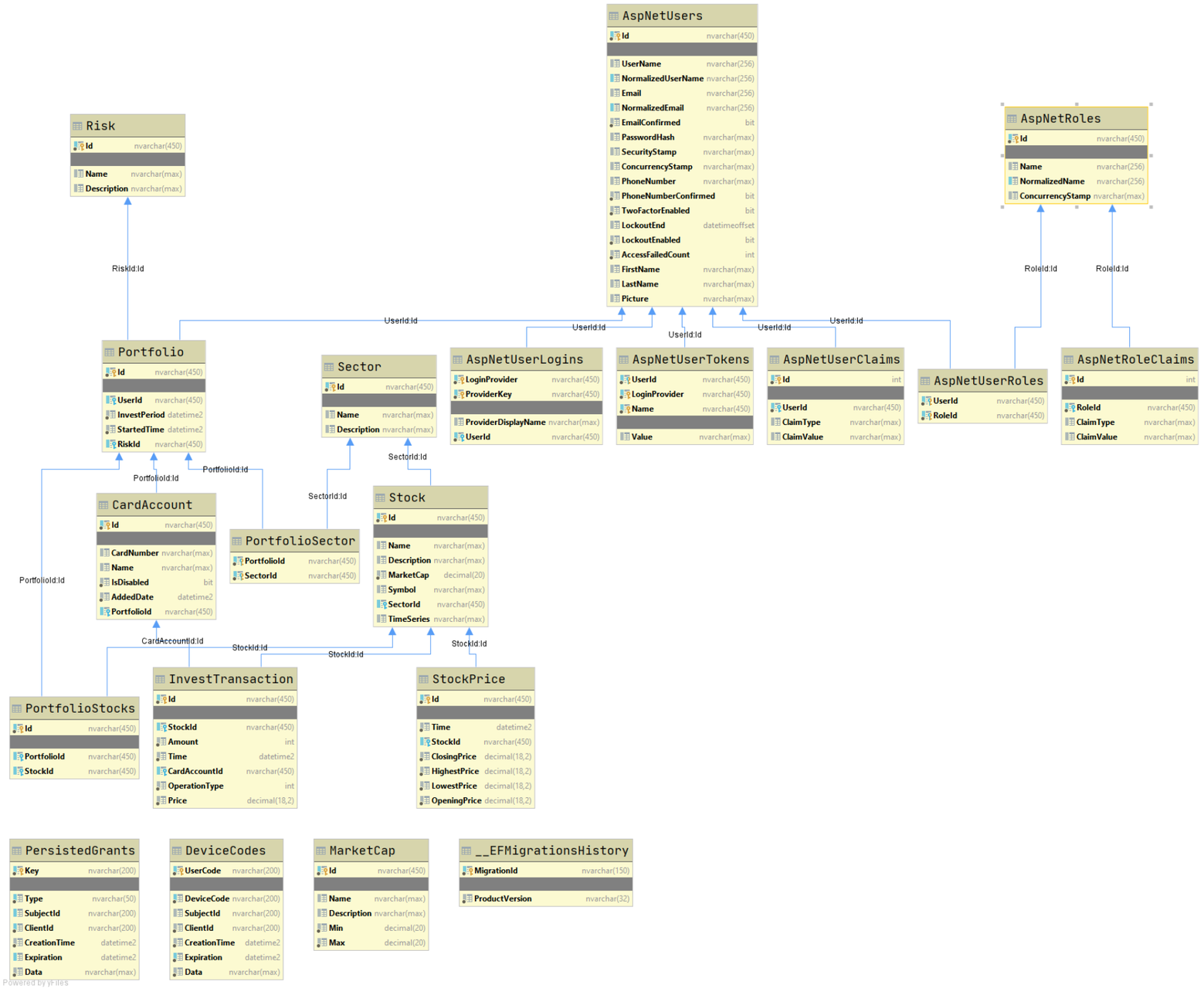
## Підміна символів

| | |
|---|---|
| Заміна символів | 4 |

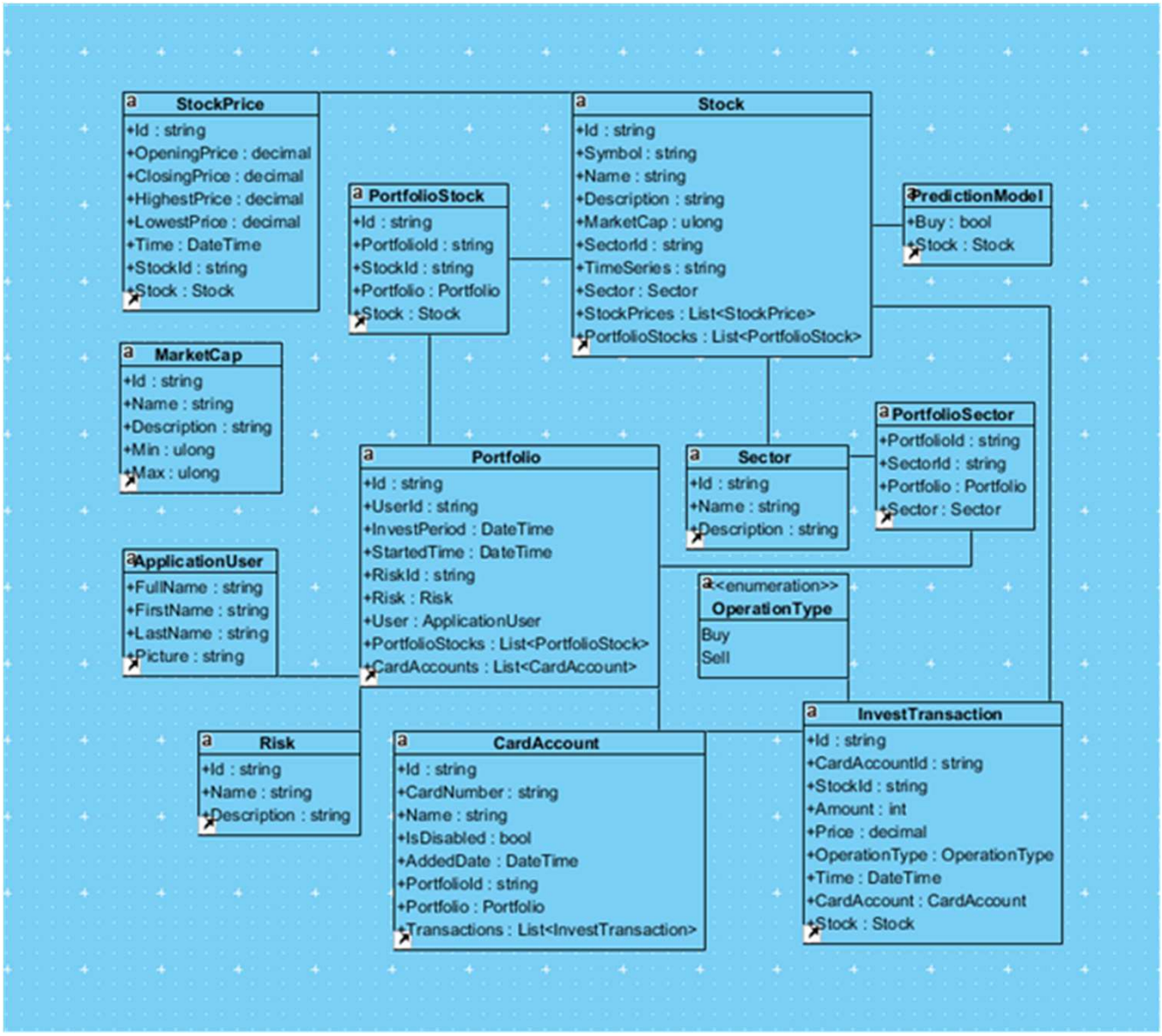
# Графічний матеріал
## до дипломного проєкту

на тему: Інформаційна система моніторингу та аналізу біржових ринків

Київ – 2020 року




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | *ДП 6123.02.000 СБД* | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| Зм. | Арк. | № документа | Підпис | Дата | *Схема бази даних* | Літера | Маса / Масштаб |
| Розробив | | Проскурка Д.М. | | | | | |
| Перевірив | | Клименко ОМ | | | | Аркуш 1 | Аркушів 1 |
| Т. кон. | | | | | | | |
| | | | | | *Інформаційна система моніторингу та аналізу біржових ринків* | *КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-41* | |
| Н. кон. | | Телишева ТО | | | | | |
| Затвердив | | Клименко ОМ | | | | | |




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ДП 6123.03.000 ССК | | |
| | | | | | | | |
| Зм. | Арк. | № документа | Підпис | Дата | Схема структурна класів програмного забезпечення | Літера | Маса / Масштаб |
| Розробив | | Проскурка ДМ | | | | | |
| Перевірив | | Клименко ОМ | | | | Аркуш 1 | Аркушів 1 |
| Т. кон. | | | | | Інформаційна система моніторингу та аналізу біржових ринків | КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61 | |
| Н. кон. | | Телишева ТО | | | | | |
| Затвердив | | Клименко ОМ | | | | | |

ДП 6123.04.000 ССП




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | *ДП 6123.04.000 ССП* | | |
| Зм. | Арк. | № документа | Підпис | Дата | *Схема структурна послідовності* | Літера | Маса / Масштаб |
| Розробив | | Проскурка ДМ | | | | | |
| Перевірив | | Клименко ОМ | | | | Аркуш 1 | Аркушів 1 |
| Т. кон. | | | | | *Інформаційна система моніторингу та аналізу біржових ринків* | *КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61* | |
| Н. кон. | | Телишева ТО | | | | | |
| Затвердив | | Клименко ОМ | | | | | |